EXHIBIT 35

UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION

VIRTAMOVE, CORP., Plaintiff, v. AMAZON.COM, INC.; AMAZON.COM SERVICES LLC; AND AMAZON WEB SERVICES, INC., Defendants.	Case No. 7:24-cv-00030 **JURY TRIAL DEMANDED**

COMPLAINT FOR PATENT INFRINGEMENT AGAINST AMAZON.COM, INC.; AMAZON.COM SERVICES LLC; AND AMAZON WEB SERVICES, INC.

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, "Plaintiff" or "VirtaMove") makes the following allegations against Defendants Amazon.com, Inc.; Amazon.com Services LLC; and Amazon Web Services, Inc. (collectively, "Defendant" or "Amazon"):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant's unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the "Asserted Patents"). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of

Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. ("Appzero"), which was established in 2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove's mission is to move and modernize the world's server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach $2.1 billion in 2019 and $4.3 billion in 2022—a compound annual growth rate ("CAGR") of 30%. *See, e.g.*, https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022. Another source reported the application containers market had a market size of $5.45 billion in 2024 and estimated it to reach $19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, https://www.mordorintelligence.com/industry-reports/application-container-market.

6. Defendant Amazon.com, Inc. is a Delaware corporation with a listed registered

agent of Corporation Service Company, 251 Little Falls Drive, Wilmington, Delaware 19808. Amazon has a principal place of business at 410 Terry Ave. North, Seattle, Washington 98109-5210. Amazon may also be served with process via its registered agent Corporation Service Company 300 Deschutes Way SW Ste 208 MC-CSC1, Tumwater, WA, 98501.

7. Defendant Amazon.com Services LLC (formerly "Amazon.com Services Inc." and referred to herein as "Amazon Services") is a limited liability company organized under the laws of the state of Delaware, with its principal place of business at 410 Terry Avenue North, Seattle, Washington 98109. Amazon Services is a wholly owned subsidiary of Amazon. Amazon Services is registered to do business in the State of Texas and may be served with process via its registered agent in Texas, Corporation Service Company dba CSC-Lawyers Incorporating Service Company at 211 7th Street, Suite 620, Austin, TX 78701-3218. Amazon Services may also be served via its Delaware registered agent Corporation Service Company, 251 Little Falls Dr., Wilmington, Delaware 19808.

8. Amazon Web Services, Inc. is a Delaware corporation with its principal place of business at 410 Terry Ave. North, Seattle, Washington 98109. Amazon Web Services, Inc. may be served through its registered agent Corporation Service Company, 211 E. 7th Street, Suite 620, Austin, Texas 78701. On information and belief, Amazon Web Services, Inc. is registered to do business in the State of Texas and has been since at least May 3, 2006. On information and belief, Amazon Web Services, LLC is a wholly-owned subsidiary of Amazon.com, Inc.

JURISDICTION AND VENUE

9. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

10. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

11. Venue is proper in this District. For example, Amazon has a regular and established place of business, including, e.g., at Amazon Tech Hub located at 11501 Alterra Parkway, Austin, Texas.

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,519,814

12. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

13. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814 patent), titled "System for Containerization of Application Sets," issued on April 14, 2009. A true and correct copy of the '814 Patent is attached as Exhibit **1.**

14. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., Amazon's AWS End-of-Support Migration Program ("EMP"), that directly infringe, literally and/or under the doctrine of equivalents, claims of the '814 patent, for example:

AWS EMP for Windows Server enables application migration from legacy Windows Server systems to supported Windows operating systems. The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system based on your requirements, and migration to an AWS environment running a supported Windows Server version. When migrated to the new server environment, the application will run on the new OS as it would on the legacy operating system. The new package redirects application requests and does not include installation of any part of the legacy operating system.

2. The application accesses data stored in a fixed location that is not available on the new OS version: the EMP engine redirects these requests to the appropriate location on the new version of the operating system.

The resulting package includes everything that the application needs to run on a modern operating system, including application files, runtimes, components, and deployment tools. The package does not include the legacy operating system, which means you never run any part of the legacy Windows Server version on the new Windows Server to which the application is upgraded.

The configuration of a package is defined in a series of XML files, which includes:

- Configurations for Registry keys and values (AppRegistry.xml)
- File, Registry, and Network redirections (Redirections.xml)
- File Type Associations (FileAssociations.xml)
- Shortcuts (Shortcuts.xml)
- Executable programs (Programs.xml)
- Environment Variables (EnvironmentVariables.xml)

The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS.

4.1.1 Application and Runtime Isolation

Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll.

4.3.1 Application to Operating System Compatibility

EMP software runs legacy applications within a compatibility package on modern, up-to-date operating systems. The EMP compatibility package does not contain any parts of the legacy operating system. Instead, it intercepts the operating system requests and redirects and resolves them against the up-to-date host operating environment.

The EMP compatibility package permits the legacy application to run alongside other applications and/or other versions of the same application. For example, multiple versions of Microsoft Office can run simultaneously on the target operating system, or two incompatible 32-bit applications can run together, isolated from each other by the EMP compatibility package.

4.3.3 Services and Drivers

EMP Compatibility packages support Windows Services out of the box. Drivers aren't captured in an EMP package but can be extracted and deployed locally using the EMP deployment script feature, as long as they are compatible with the target operating system.

https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf.

15. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

16. Defendant's infringement has been and is willful. Defendant knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2013. For example, on or about October 2013, in connection with the prosecution of U.S. Patent No. 8,806,655 (assigned to Amazon), the examiner cited the '814 Patent against Amazon. On or about June 2020, in connection with the prosecution of U.S. Patent No. 11,061,812 (assigned to Amazon), the examiner cited U.S. Pub. No. 2005/0060722 (which issued as the '814 Patent) against Amazon. Defendant knew, or should have known, that its conduct amounted to infringement of the '814 patent. Accordingly, Defendant is liable for willful infringement.

17. Defendant also knowingly and intentionally induces infringement of claims of the '814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and/or earlier, as set forth above. Despite this knowledge of the '814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through

user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '814 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '814 patent, thereby specifically intending for and inducing its customers to infringe the '814 patent through the customers' normal and customary use of the Accused Products.

18. Defendant has also infringed, and continue to infringe, claims of the '814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in violation of 35 U.S.C. § 271(c).

19. The Accused Products satisfy all claim limitations of claims of the '814 patent. A claim chart comparing an independent claim of the '814 patent to a representative Accused Product, is attached as Exhibit **2**, which is hereby incorporated by reference in its entirety.

20. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814 patent pursuant to 35 U.S.C. § 271.

21. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with

interest and costs as fixed by the Court.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

22. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

23. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as Exhibit **3**.

24. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., Amazon's AWS Elastic Container Service ("ECS"), that directly infringe, literally and/or under the doctrine of equivalents, claims of the '058 patent, for example:

Amazon ECS supports using 64-bit ARM applications. You can run your applications on the platform that's powered by AWS Graviton2 processors,. It's suitable for a wide variety of workloads. This includes workloads such as application servers, micro-services, high-performance computing, CPU-based machine learning inference, video encoding, electronic design automation, gaming, open-source databases, and in-memory caches.

An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into an Amazon ECS cluster. When you run tasks with Amazon ECS using the EC2 launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances.

Note

Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply.

You can use EC2 instances with the following Linux operating systems to run your applications:

- Amazon Linux: This is a general purpose operating system.
- Bottlerocket: Bottlerocket is a Linux based open-source operating system that is purpose built by AWS for running containers on virtual machines or bare metal hosts. The Amazon ECS-optimized Bottlerocket AMI is secure and only includes the minimum number of packages that's required to run containers. This improves resource usage, reduces security attack surface, and helps lower management overhead. For information about the security features and guidance, see Security Features and Security Guidance on the GitHub website.

An Amazon ECS container instance specification consists of the following components:

Required

- A Linux distribution running at least version 3.10 of the Linux kernel.

- The Amazon ECS container agent (preferably the latest version). For more information, see Updating the Amazon ECS container agent.

You must architect your applications so that they can run on *containers*. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an *image*. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a *registry* such as Amazon ECR where they can be downloaded from.

Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see Docker Compose CLI - Amazon ECS.

https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf.

25. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

26. Defendant also knowingly and intentionally induces infringement of claims of the ’058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the ’058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed. Despite this knowledge of the ’058 patent, Defendant continues to actively encourage and

instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

27. Defendant has also infringed, and continue to infringe, claims of the '058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in violation of 35 U.S.C. § 271(c).

28. The Accused Products satisfy all claim limitations of claims of the '058 patent. A claim chart comparing an independent claim of the '058 patent to a representative Accused Product, is attached as Exhibit **4**, which is hereby incorporated by reference in its entirety.

29. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

30. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no

event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;

b. A judgment in favor of Plaintiff that Defendant has willfully infringed the '814 patent;

c. A permanent injunction prohibiting Defendant from further acts of infringement of each of the '814 and '058 patents;

d. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant's infringement of each of the Asserted Patents;

e. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;

f. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys' fees against Defendant; and

g. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury

of any issues so triable by right.

Dated: January 26, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)
rmirzaie@raklaw.com
Marc A. Fenster (CA SBN 181067)
mfenster@raklaw.com
Neil A. Rubin (CA SBN 250761)
nrubin@raklaw.com
Amy E. Hayden (CA SBN 287026)
ahayden@raklaw.com
Christian W. Conkle (CA SBN 306374)
cconkle@raklaw.com
Jonathan Ma (CA SBN 312773)
jma@raklaw.com

RUSS AUGUST & KABAT
12424 Wilshire Boulevard, 12th Floor
Los Angeles, CA 90025
Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

Exhibit 1

US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.: US 7,519,814 B2**
(45) **Date of Patent: Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248; 709/214; 719/319

(58) **Field of Classification Search** 713/167; 719/319
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

WO WO 2006/039181 A 4/2006

OTHER PUBLICATIONS

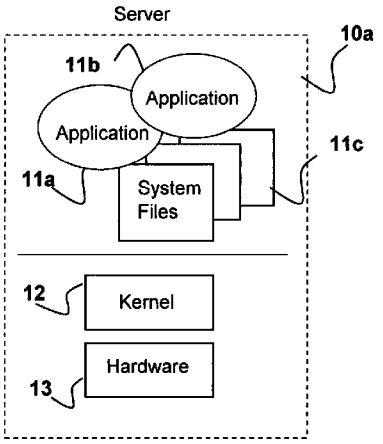
Soltesz, S., et al, 'Container-based Operating System Virtualization:A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697& CFTOKEN=4667.*

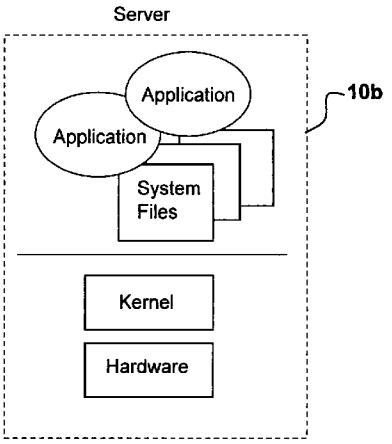
Primary Examiner—Kambiz Zand
Assistant Examiner—Ronald Baum
(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets





U.S. PATENT DOCUMENTS

6,847,970	B2 *	1/2005	Keller et al.	707/100
7,076,784	B1 *	7/2006	Russell et al.	719/315
7,287,259	B2 *	10/2007	Grier et al.	719/331
2002/0004854	A1	1/2002	Hartley	
2002/0174215	A1	11/2002	Schaefer	709/224
2003/0101292	A1	5/2003	Fisher	

* cited by examiner

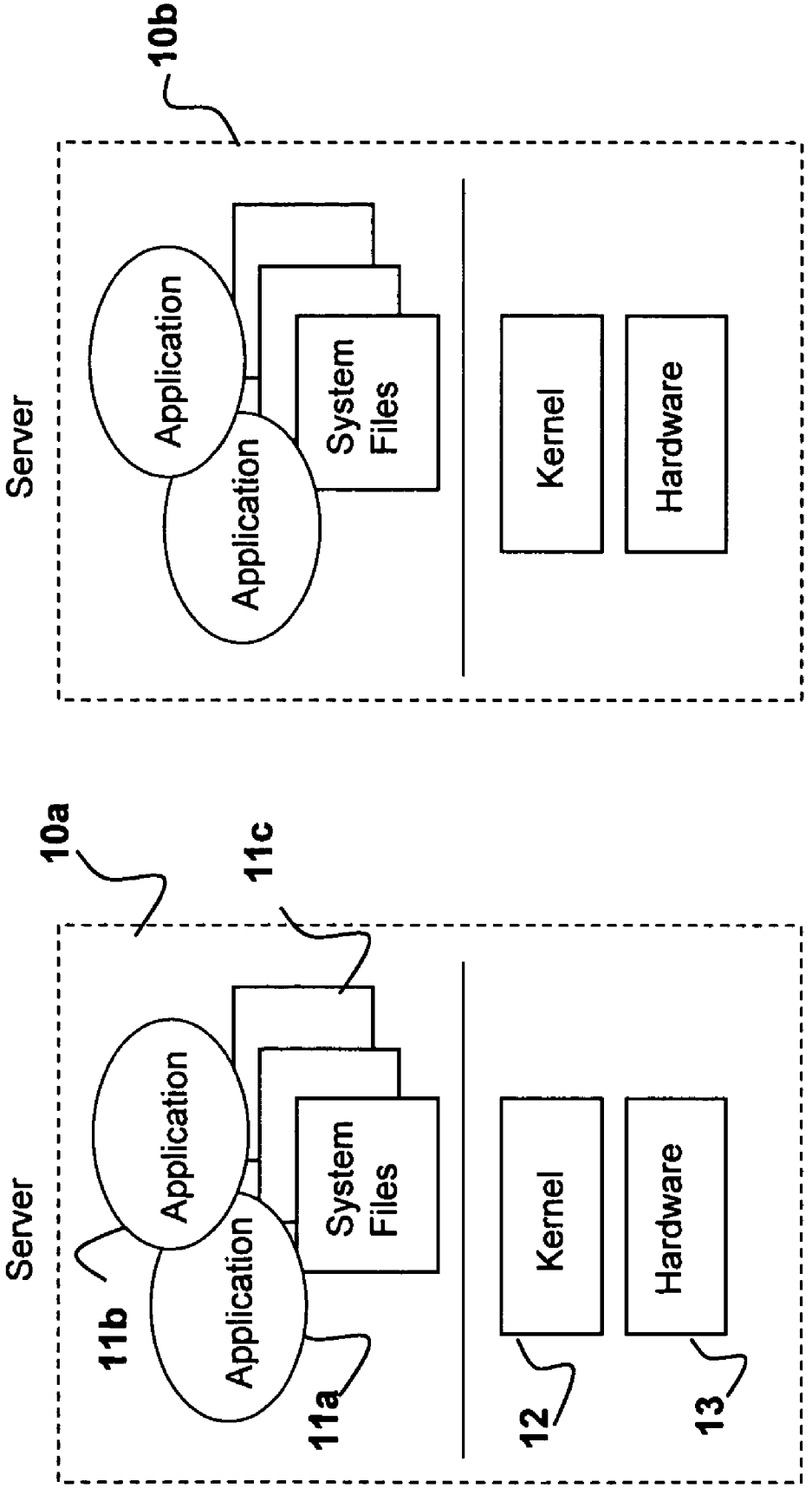

10b
Server
Application
Application
System Files
Kernel
Hardware
10a
11c
Server
Application
Application
System Files
Kernel
Hardware
11b
11a
12
13

Figure 1

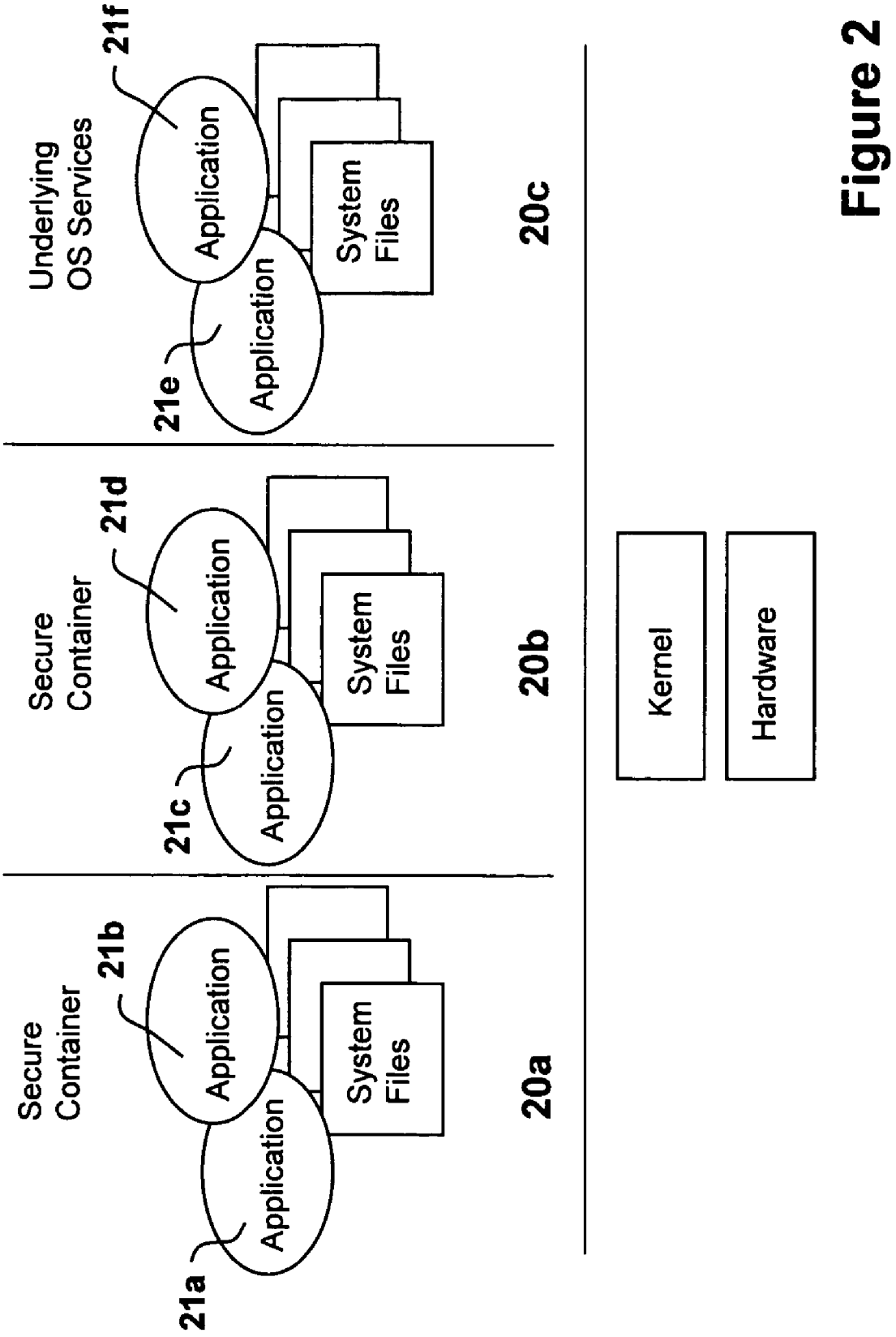

Underlying OS Services
21f
Application
Application
System Files
21e
20c
Secure Container
21d
Application
Application
System Files
21c
20b
Secure Container
21b
Application
Application
System Files
21a
20a
Kernel
Hardware

Figure 2

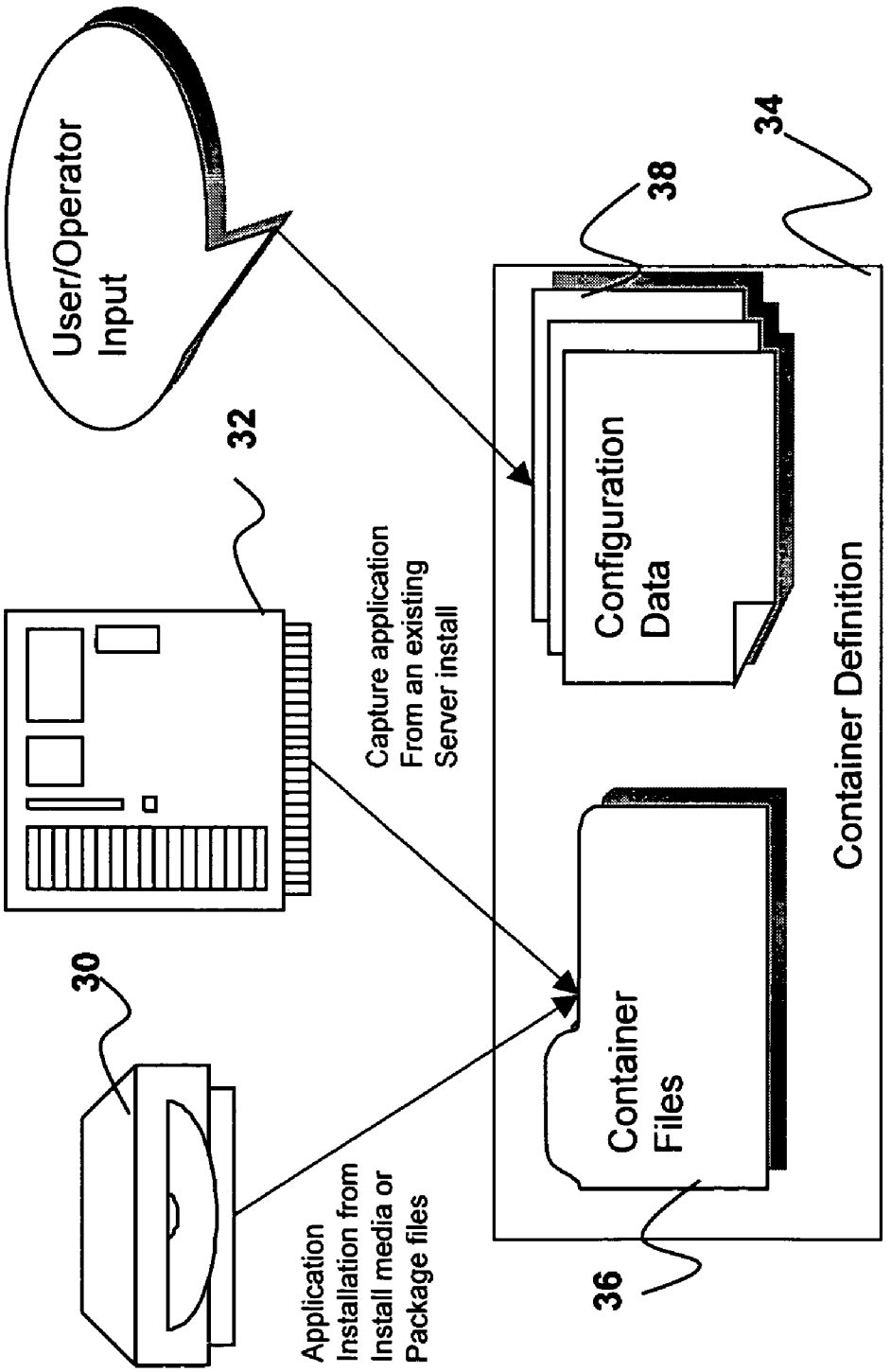

User/Operator Input
32
34
38
Configuration Data
Capture application From an existing Server install
Container Definition
30
Container Files
36
Application Installation from Install media or Package files

Figure 3

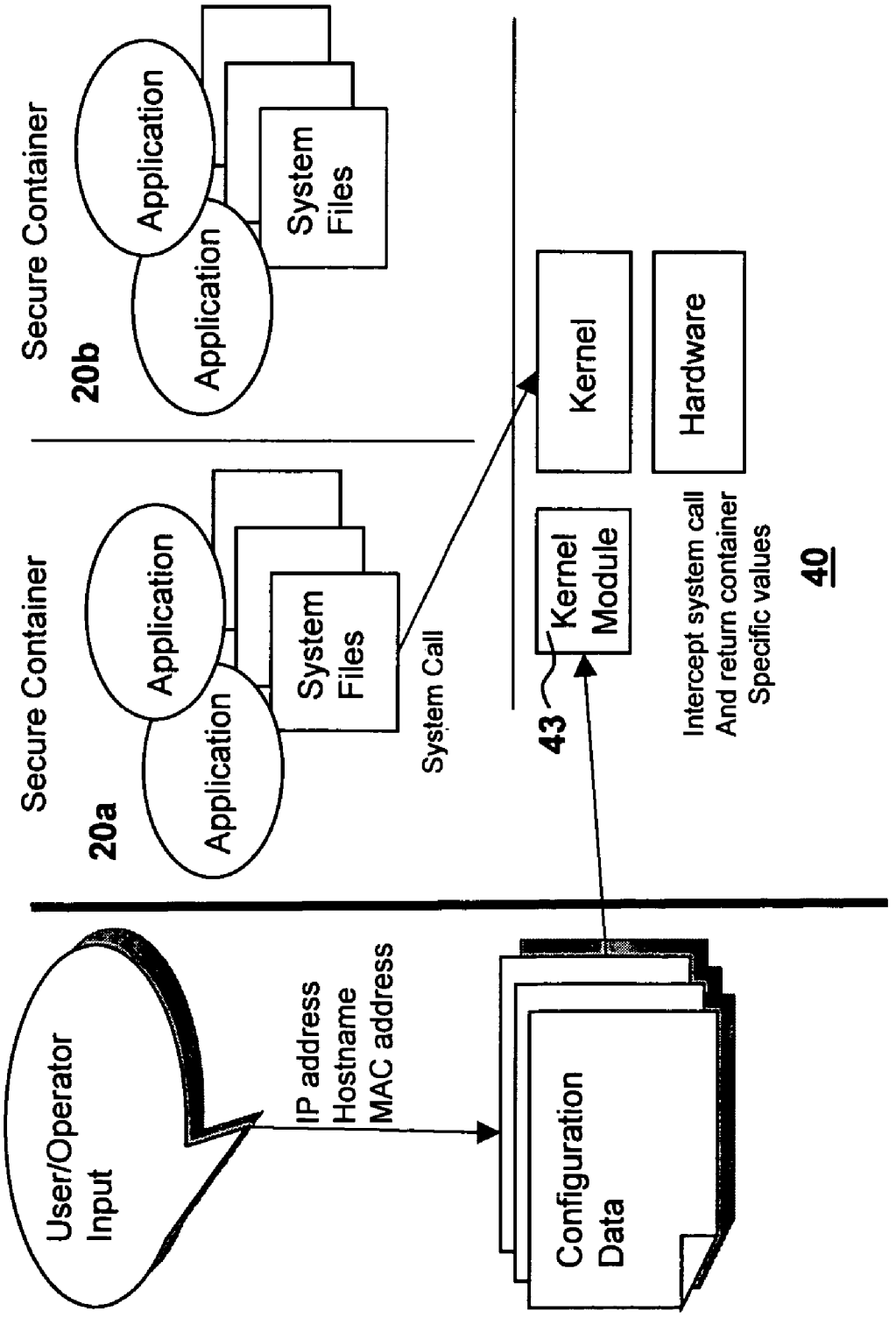

Secure Container
20b
Application
Application
System Files
Secure Container
20a
Application
Application
System Files
System Call
Kernel
Hardware
Kernel Module
43
Intercept system call
And return container
Specific values
40
User/Operator Input
IP address
Hostname
MAC address
Configuration Data

Figure 4

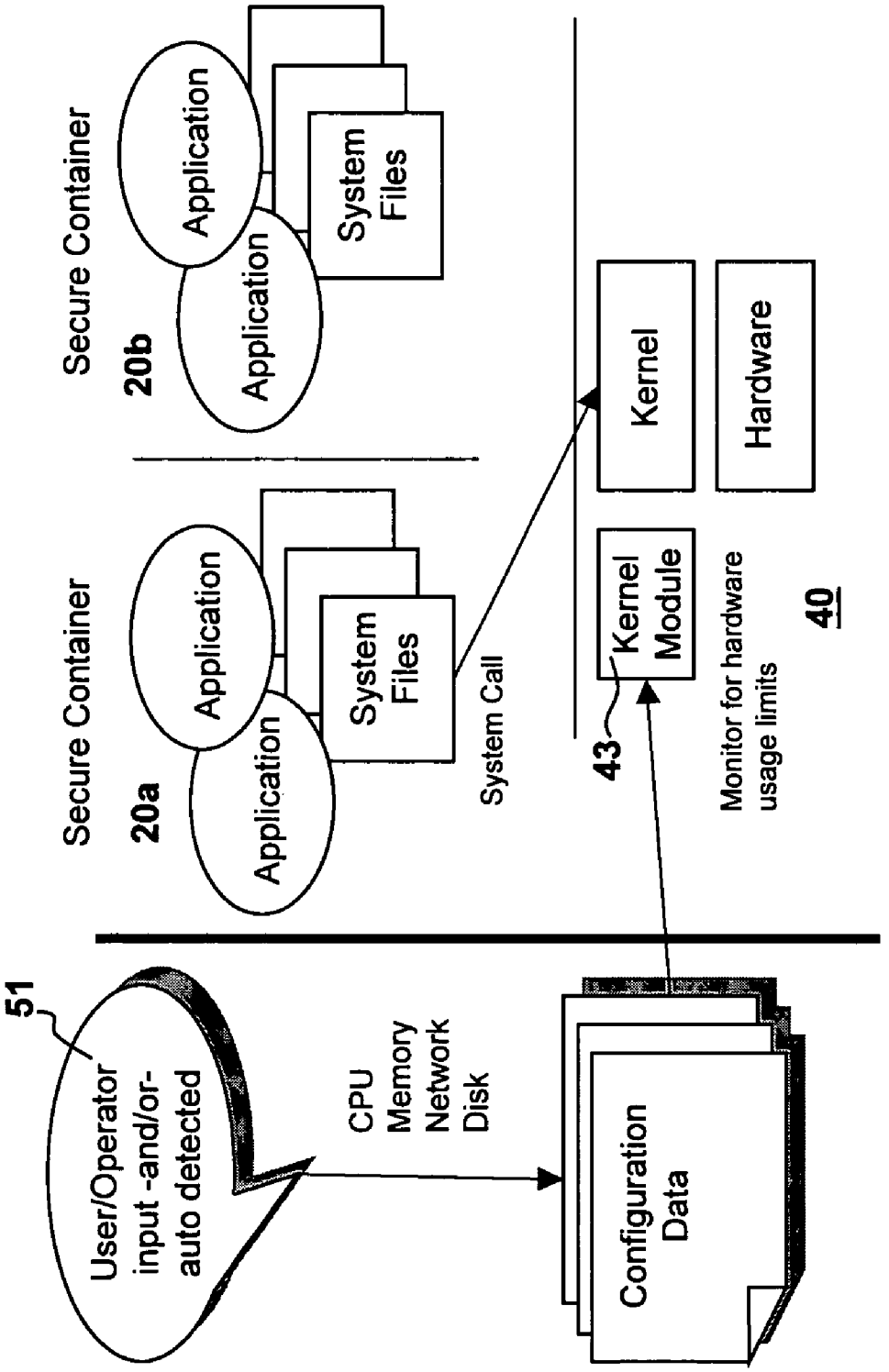

Secure Container
20b
Application
Application
System Files
Secure Container
20a
Application
Application
System Files
System Call
Kernel
Hardware
Kernel Module
43
Monitor for hardware usage limits
40
51
User/Operator input -and/or- auto detected
CPU
Memory
Network
Disk
Configuration Data

Figure 5

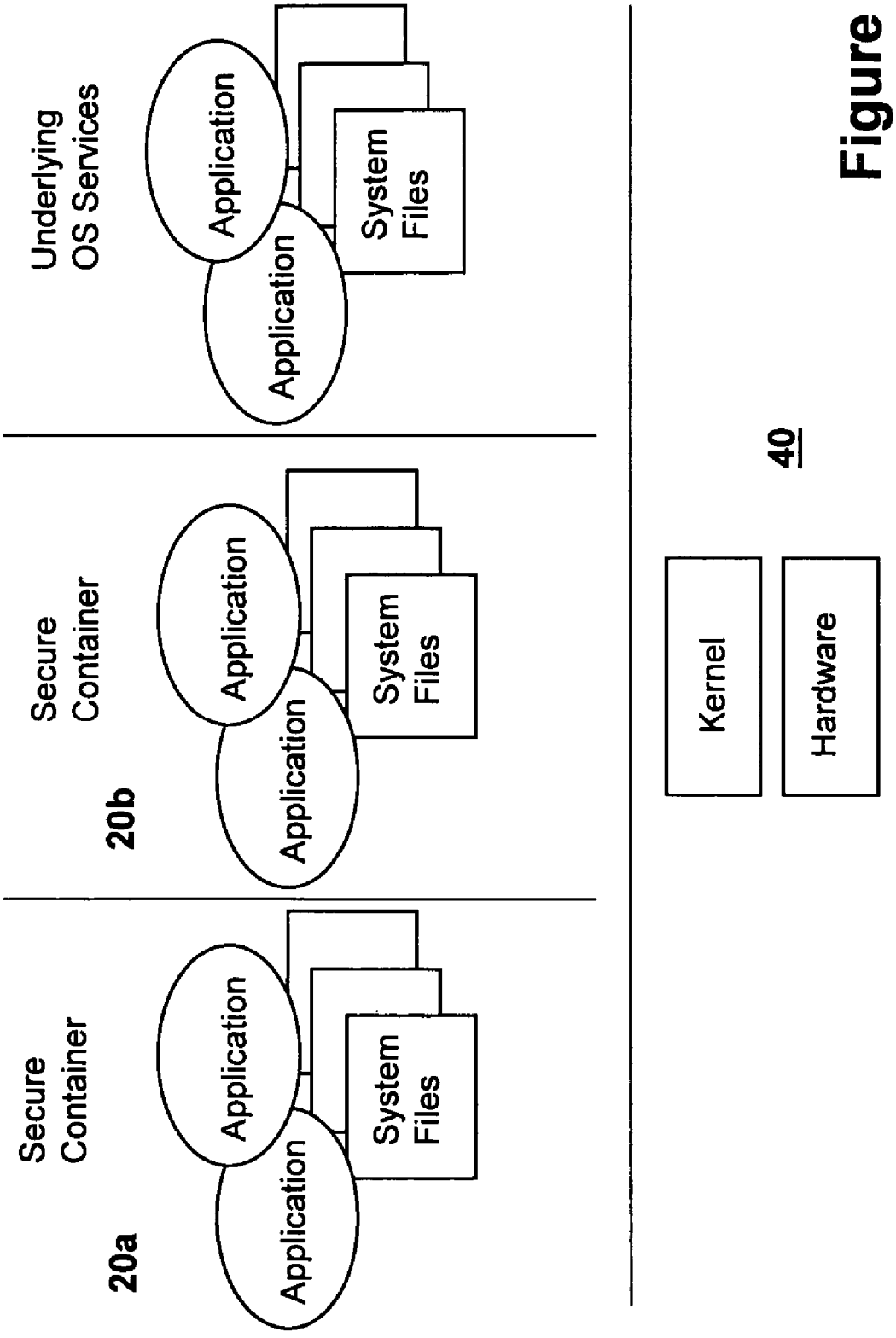

Underlying OS Services
Application
System Files
Application
Secure Container
20b
Application
System Files
Application
Secure Container
20a
Application
System Files
Application
Kernel
Hardware
40

Figure 6

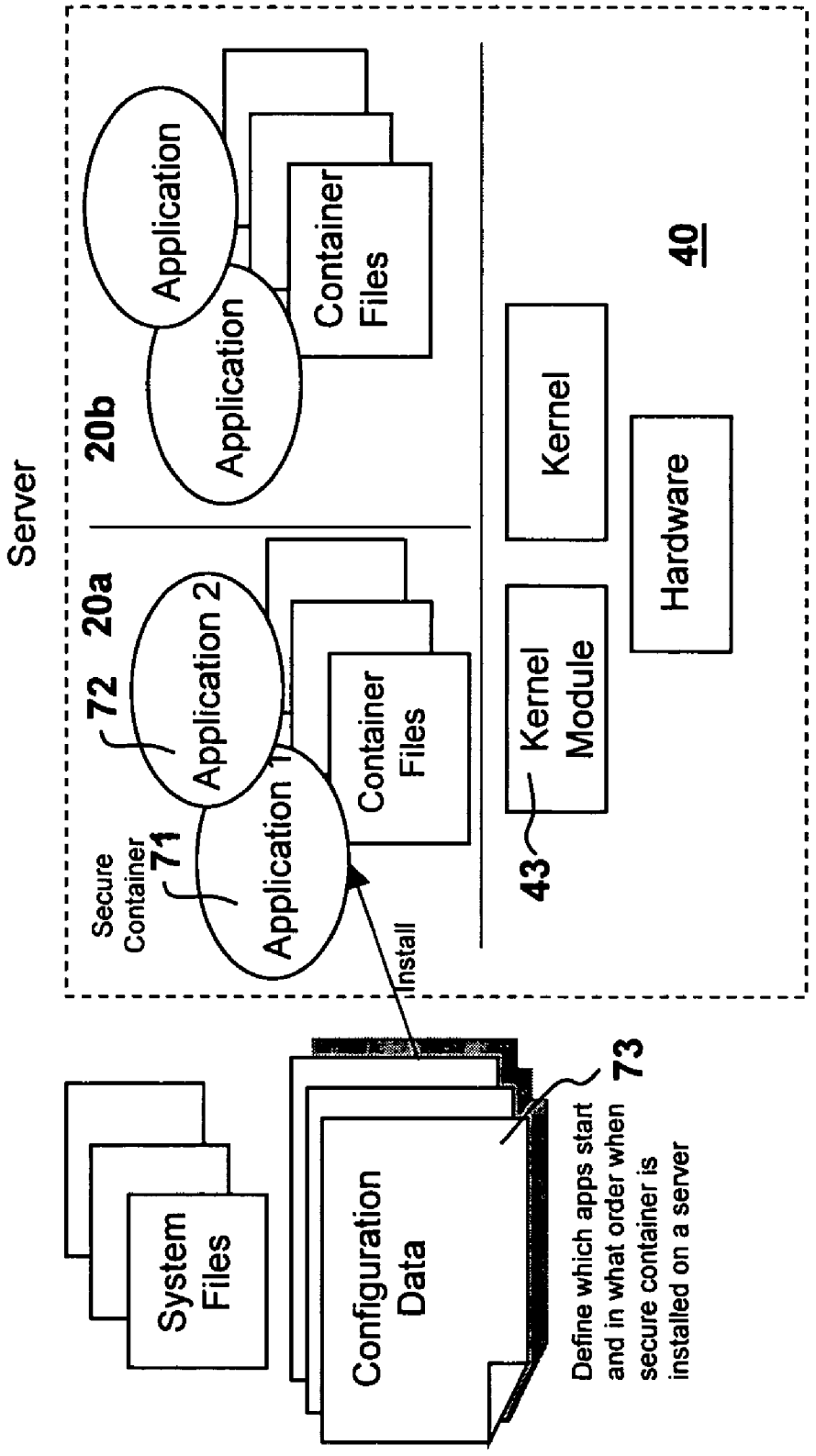

Server
20b
Application
Application
Container Files
Kernel
40
Hardware
20a
72
Application 2
Application 1
Container Files
Secure Container
71
Kernel Module
43
Install
System Files
Configuration Data
Define which apps start and in what order when secure container is installed on a server
73

Figure 7

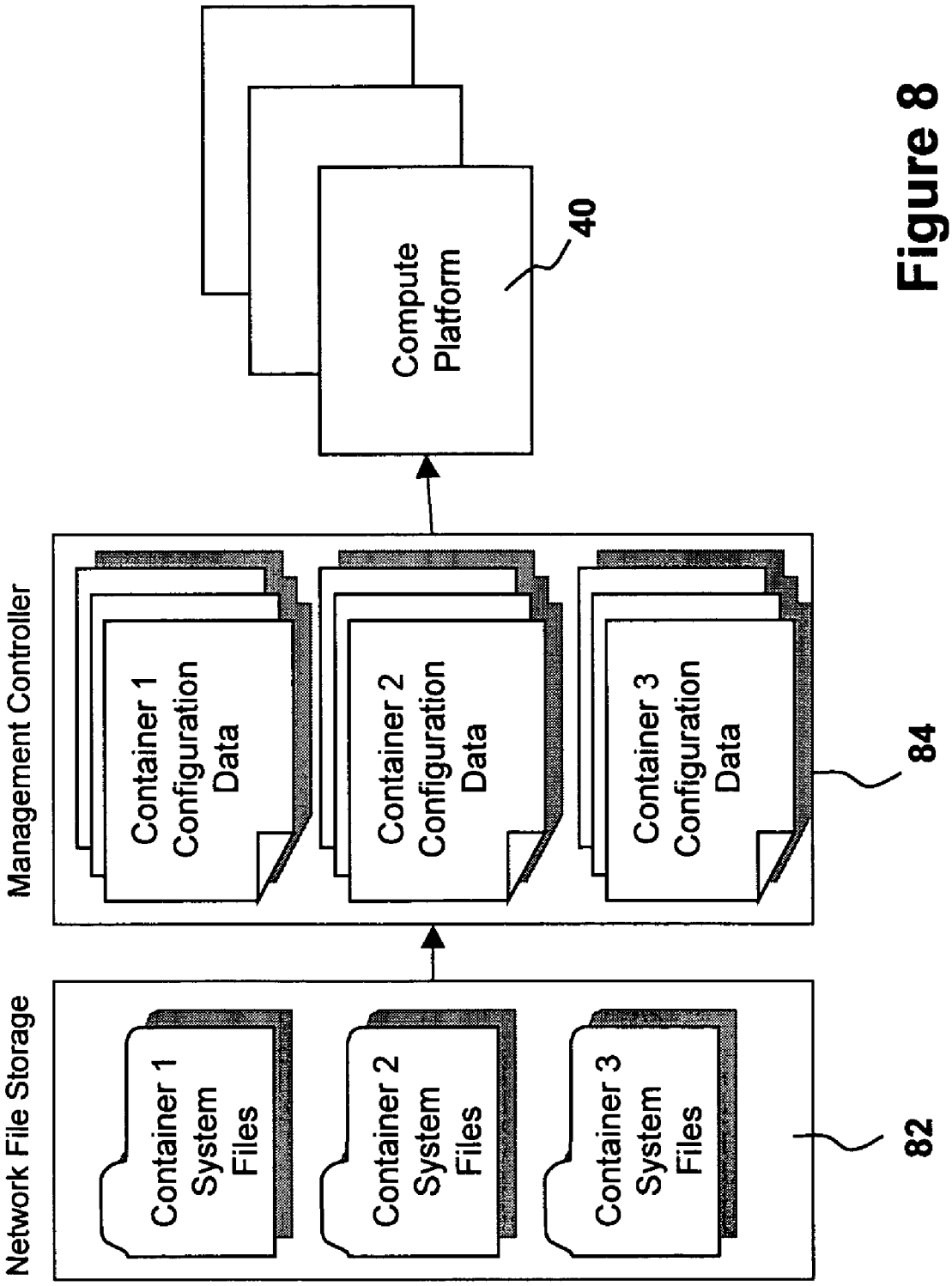

Compute Platform
40
Management Controller
Container 1 Configuration Data
Container 2 Configuration Data
Container 3 Configuration Data
84
Network File Storage
Container 1 System Files
Container 2 System Files
Container 3 System Files
82

Figure 8

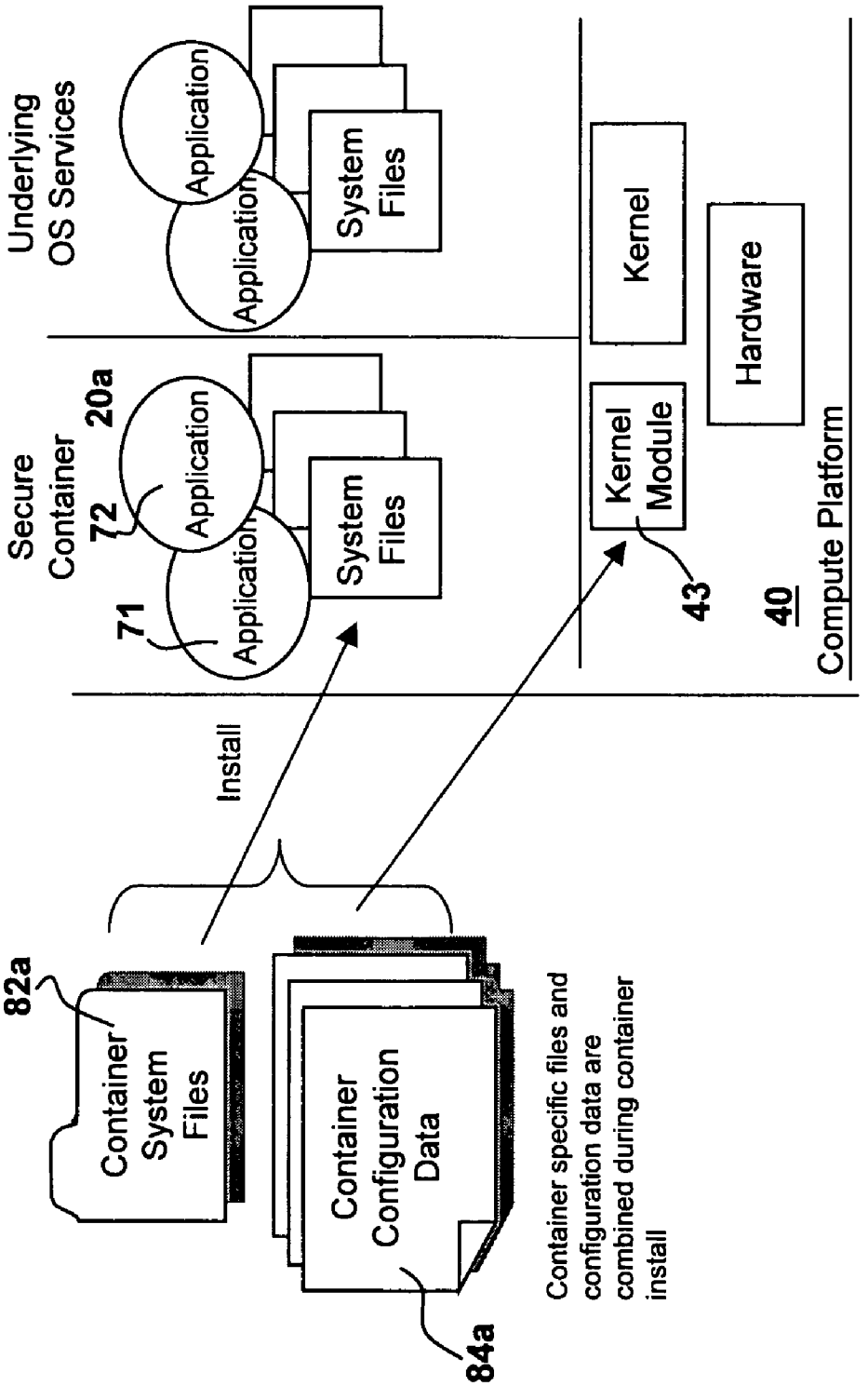

Underlying OS Services
Application
Application
System Files
Secure Container
20a
72
71
Application
Application
System Files
Kernel
Kernel Module
Hardware
43
40
Compute Platform
Install
82a
Container System Files
Container Configuration Data
84a
Container specific files and configuration data are combined during container install

Figure 9

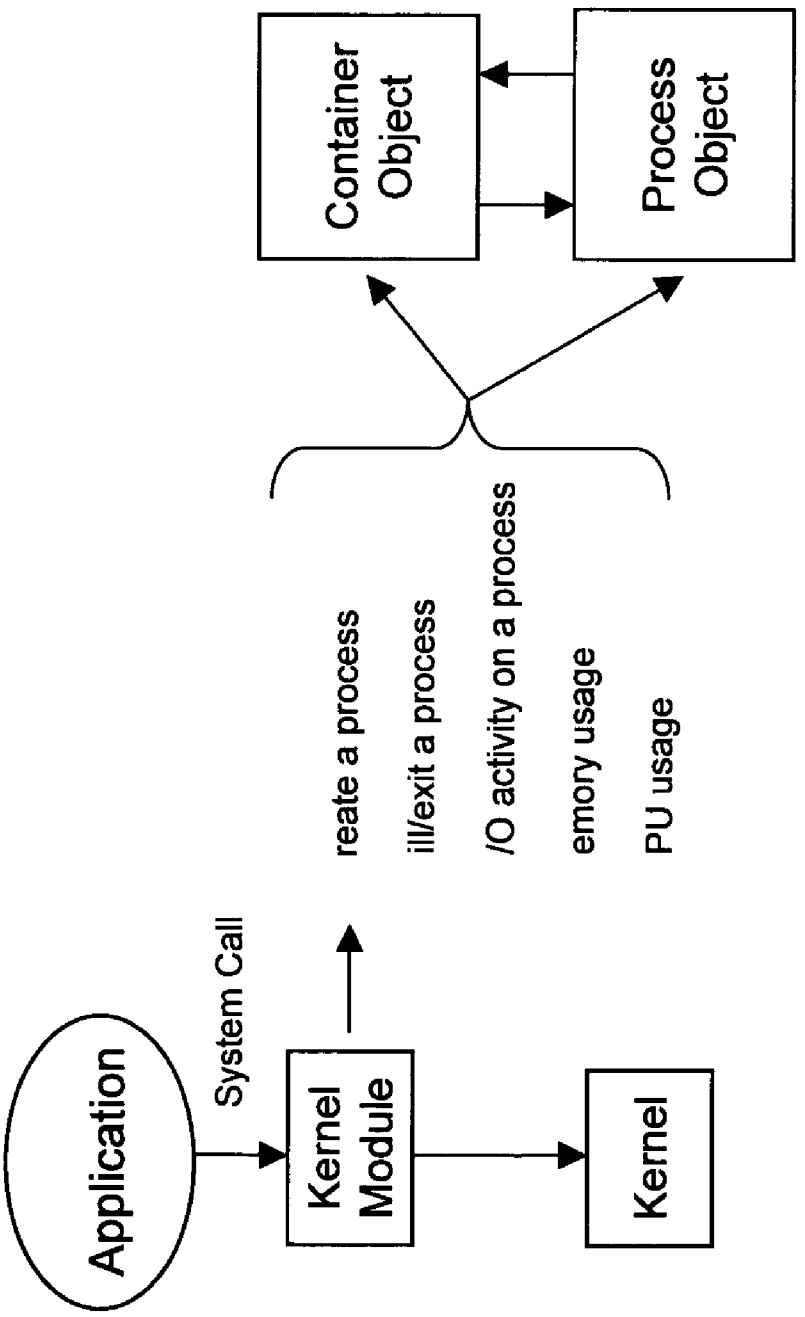

Container Object
Process Object
reate a process
ill/exit a process
/O activity on a process
emory usage
PU usage
Application
System Call
Kernel Module
Kernel

Figure 10

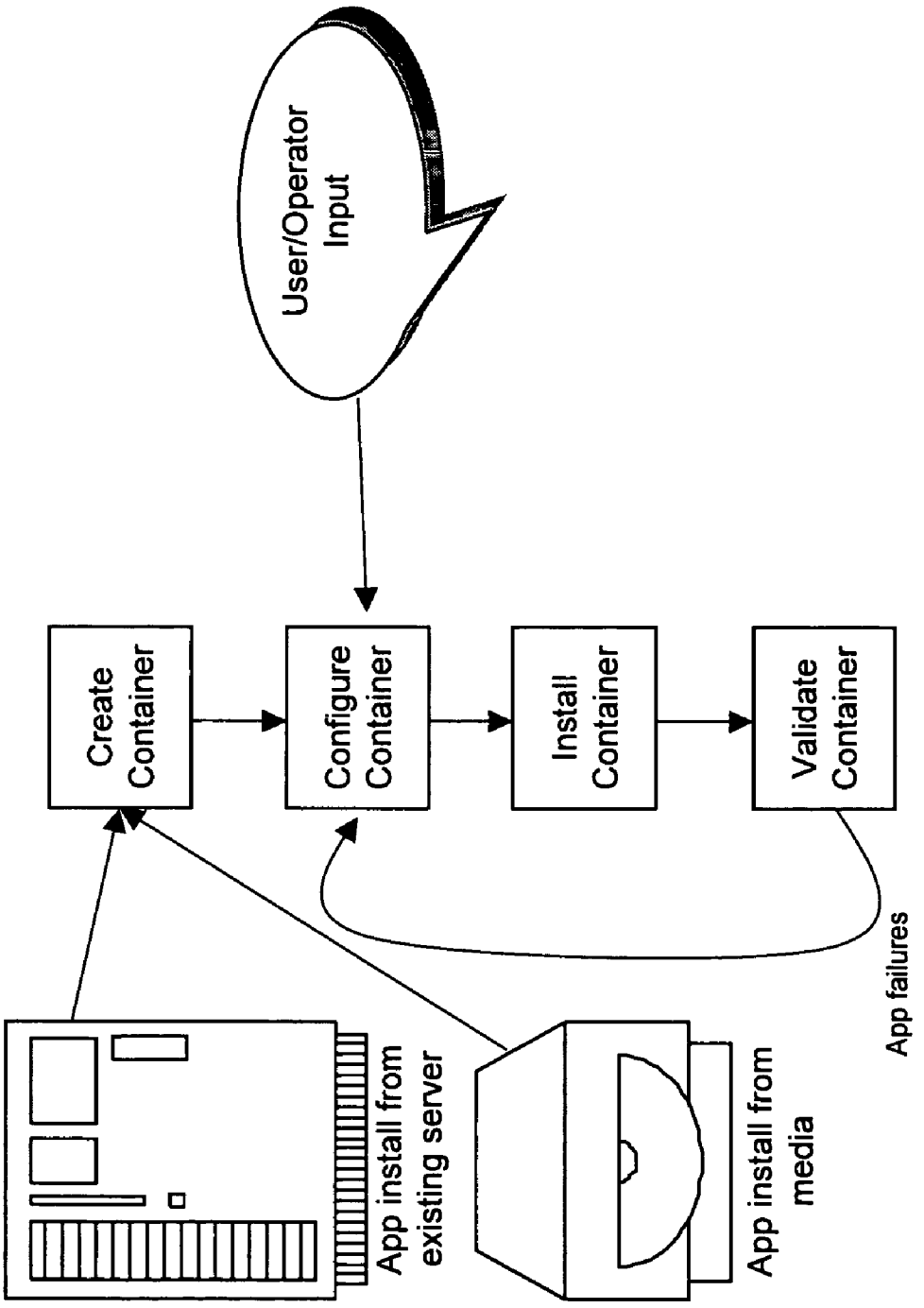

User/Operator Input
Create Container
Configure Container
Install Container
Validate Container
App failures
App install from existing server
App install from media

Figure 11

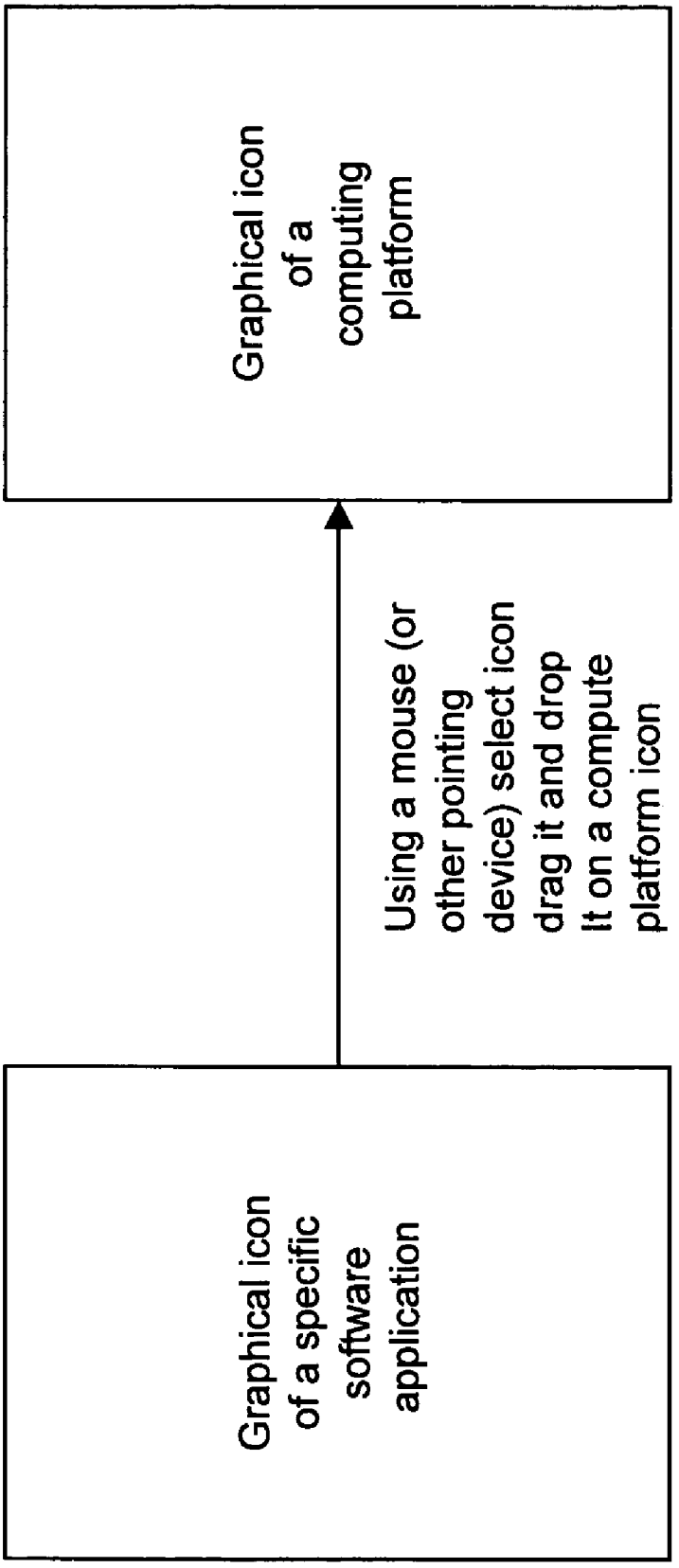

Graphical icon of a computing platform
Using a mouse (or other pointing device) select icon drag it and drop It on a compute platform icon
Graphical icon of a specific software application

Figure 12

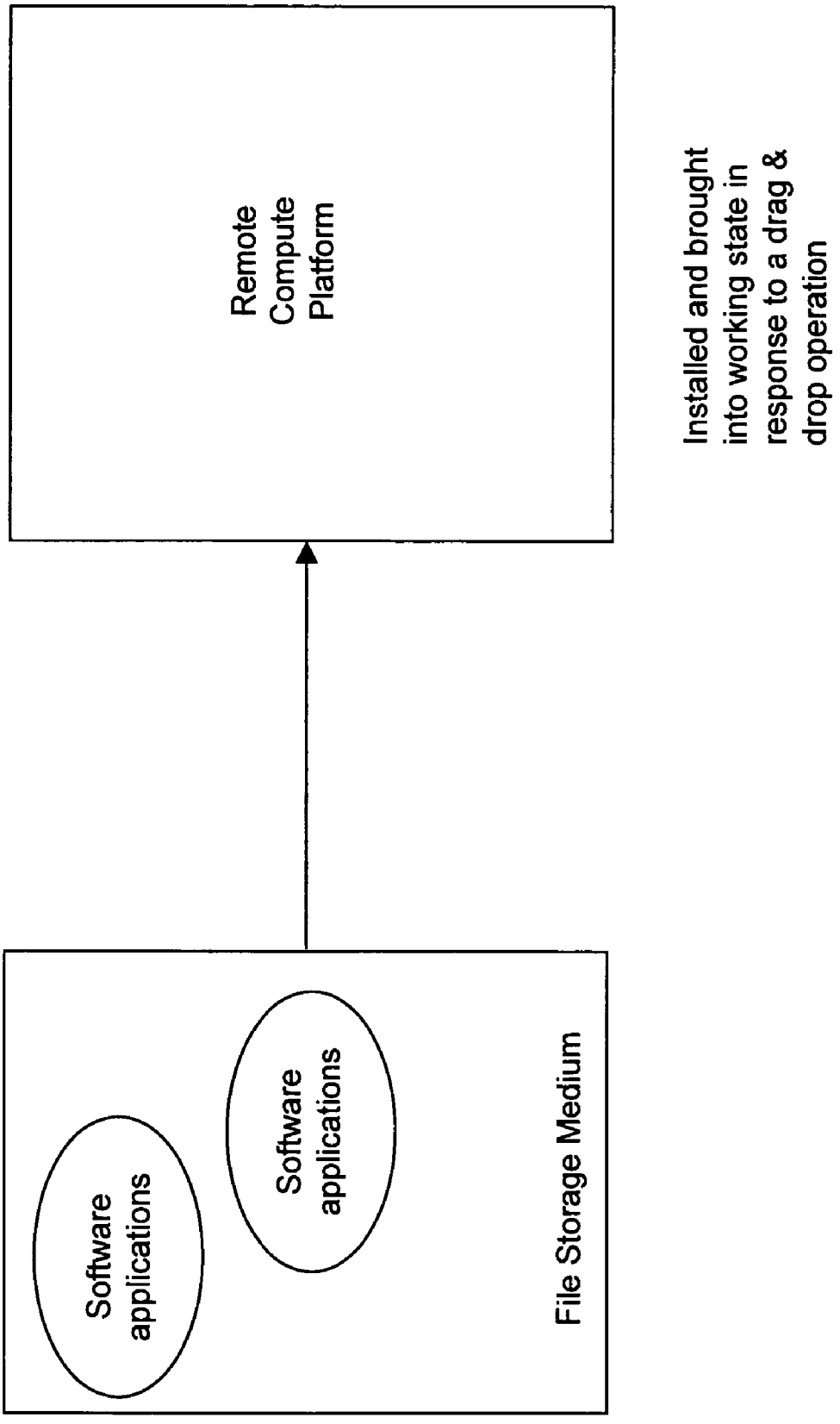

Remote Compute Platform
Installed and brought into working state in response to a drag & drop operation
Software applications
Software applications
File Storage Medium

Figure 13

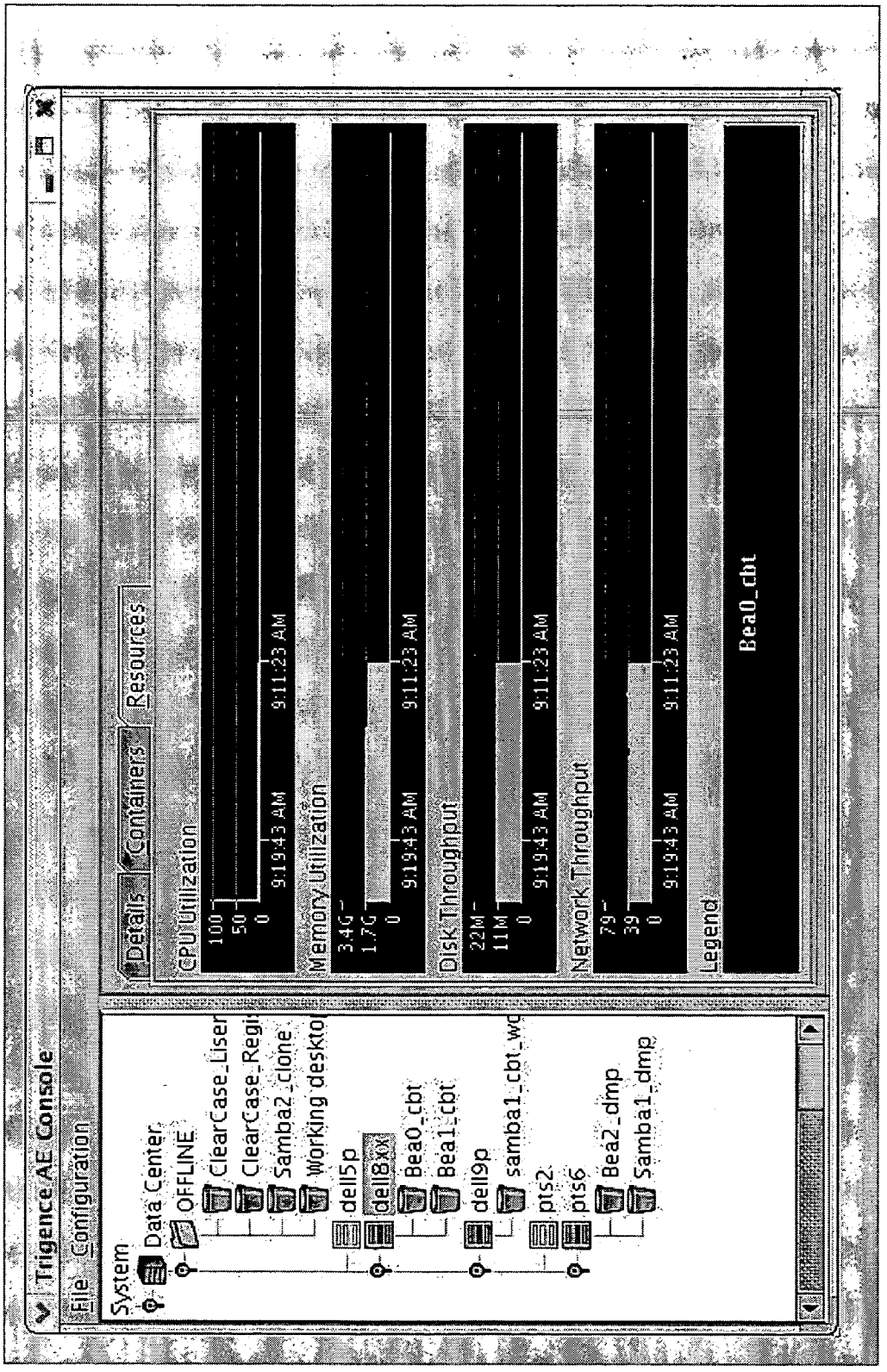

Trigence AE Console
File Configuration
System
Data Center
OFFLINE
ClearCase_Liser
ClearCase_Regi
Samba2_clone
Working deskto
dell5p
dell8xx
Bea0_cbt
Bea1_cbt
dell9p
samba1_cbt_wo
pts2
pts6
Bea2_dmp
Samba1_dmp
Details
Containers
Resources
CPU Utilization
100
50
0
9:19:43 AM
9:11:23 AM
Memory Utilization
3.4G
1.7G
0
9:19:43 AM
9:11:23 AM
Disk Throughput
22M
11M
0
9:19:43 AM
9:11:23 AM
Network Throughput
79
39
0
9:19:43 AM
9:11:23 AM
Legend
Bea0_cbt

Figure 14

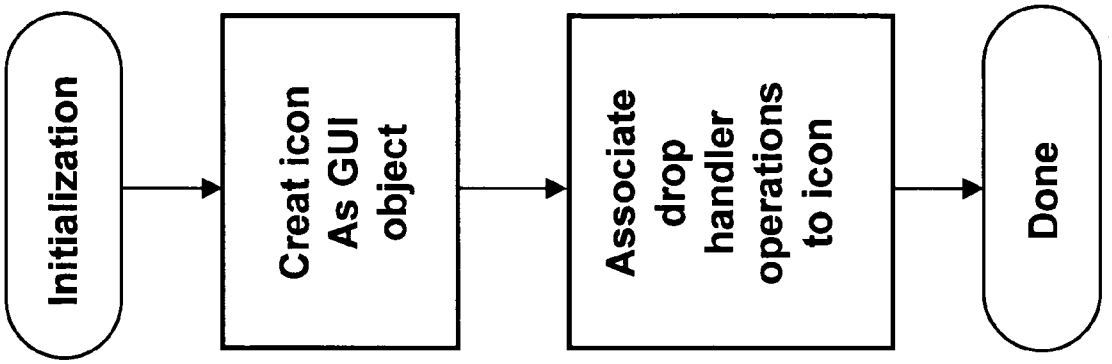

Initialization
Creat icon As GUI object
Associate drop handler operations to icon
Done

Figure 15

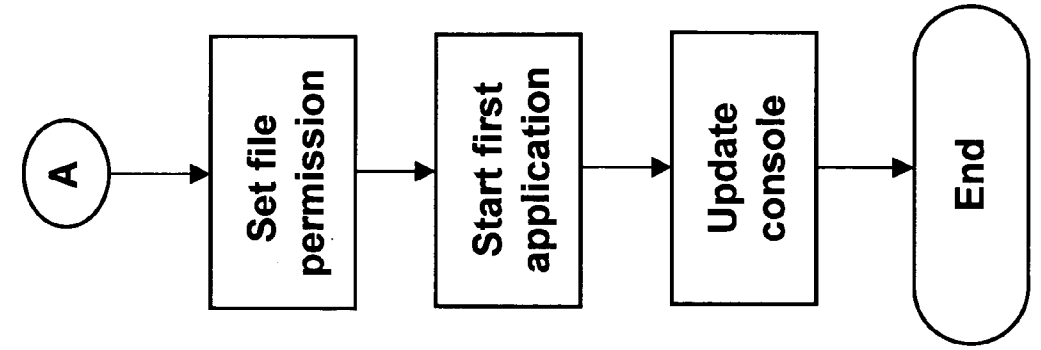
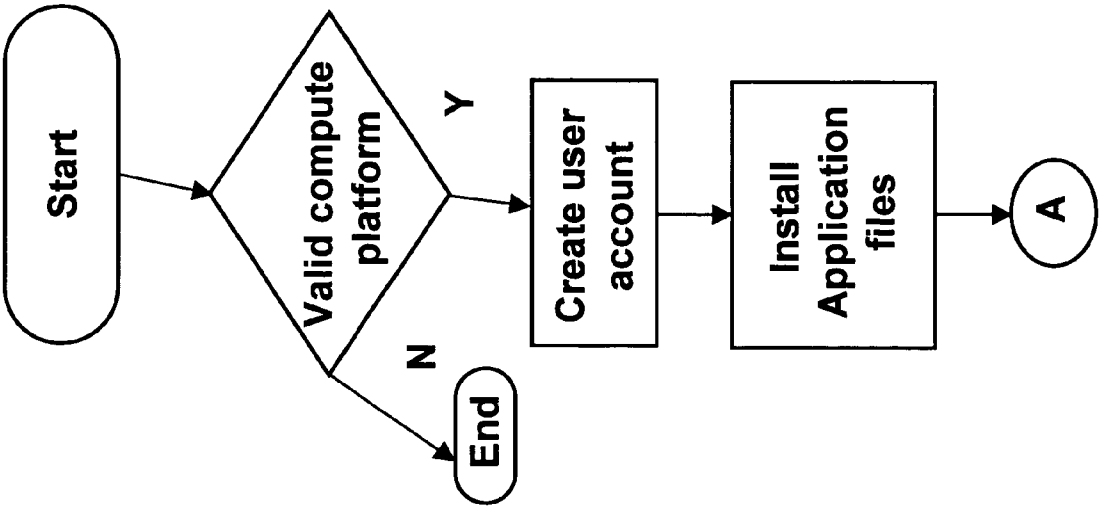
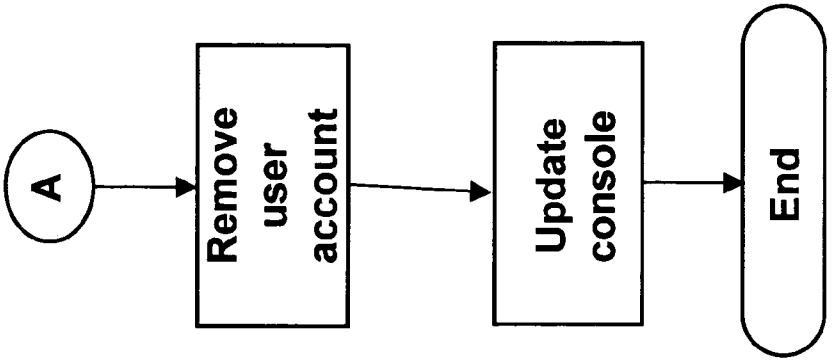

A
Set file permission
Start first application
Update console
End

Figure 16


Start
Valid compute platform
Y
N
Create user account
Install Application files
A
End


A
Remove user account
Update console
End

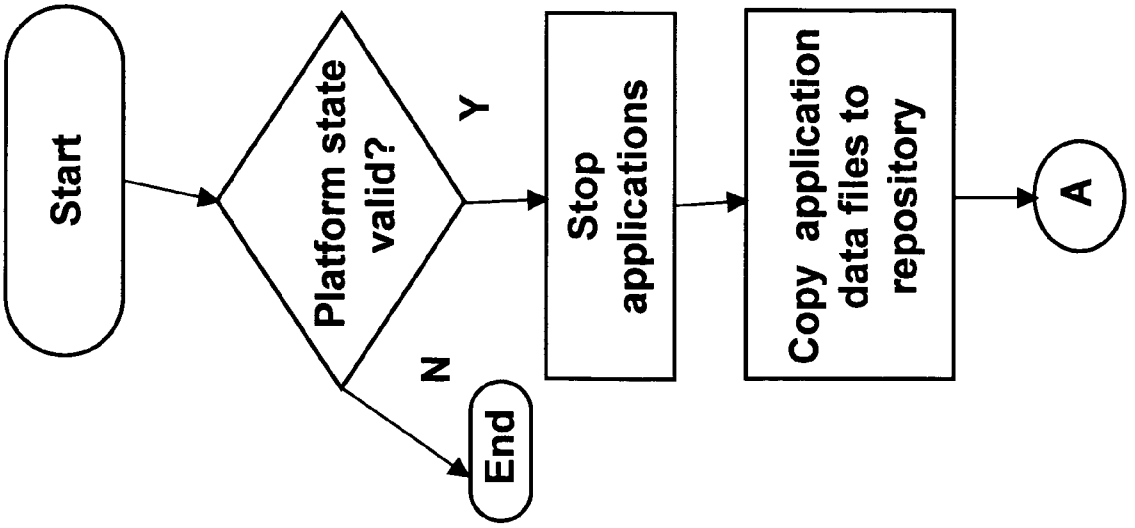

Start
Platform state valid?
Y
N
End
Stop applications
Copy application data files to repository
A

Figure 17

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1

/lib/libnsl.so.1
/lib/libdl.so.2
/lib/i686/libpthread.so.0
/lib/i686/libc.so.6
/lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
/etc/httpd/conf
/etc/httpd/conf.d
/etc/httpd/logs
/etc/httpd/modules
/etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySql and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed "spoofing" and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. **1** is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. **2** is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. **3** is a pictorial diagram illustrating the creation of a container.

FIG. **4** is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the "kernel module" which is used to handle system calls.

FIG. **5** is a diagram similar to FIG. **4**, wherein additional configuration data is provided.

FIG. **6** is a diagram illustrating a system wherein the containers are secure containers.

FIG. **7** is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. **8** is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. **9** is a diagram that illustrates the installation of a container on a server.

FIG. **10** is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. **11** is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. **12** is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. **13** is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. **14** is a screen snapshot of an implementation according to the schematic of FIG. **13**.

FIG. **15** is a flow chart of a method of initializing icons of FIG. **14**.

FIG. **16** is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. **14**; and,

FIG. **17** is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. **13**.

DETAILED DESCRIPTION

Turning now to FIG. **1**, a system is shown having a plurality of servers **10***a*, **10***b*. Each server includes a processor and an independent operating system. Each operating system includes a kernel **12**, hardware **13** in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs **11***a*, **11***b* and related system files **11***c*.

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. **2** illustrates a system in accordance with the invention in which multiple applications **21***a*, **21***b*, **21***c*, **21***d*, **21***e*, and **21***f* can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20***a*, **20***b* and **20***c*. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20***a*, **20***b* and **20***c* are shown in FIG. **2** in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. **3** the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. **4** illustrates a system having two secure containers **20***a* **20***b*, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20***a*, **20***b* environment system calls are intercepted, by the kernel module **43** passing through services installed as part of this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by "spoofing" the system.

As introduced heretofore, "spoofing" is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system's kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the 'uname' system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has 'spoofed' the 'uname' system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same "spoofing" mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

tracking applications, for example a tracking of which applications are in and out of a specific container;

spoofing particular values returned by the kernel;

applying resource checks and imposing resource limits;

monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,

retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. **5** including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20***a* or **20***b*. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. **6** that each application executing associated with a container **20***a* or **20***b*, or contained within a

container **20***a* or **20***b*, is able to accesses files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20***a* association are not able to access the files contained in another container **20***b*.

When a container **20***a* or **20***b* is installed specific applications are started, as is shown in FIG. **7** and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. **7**, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. **8**, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. **9** shows that the files **82***a*, **84***a* are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. **9**.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. **10** shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. **3** illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform an then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. **3**, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. **3**. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

modifying the contents a file to add a container specific IP address;

modifying the contents of a directory to define start scripts that should be executed;

modifying the contents of a file to define which mount points will relate to a container;

removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is “dropped” on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. **12**, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. **13**, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. **14**, a screen snapshot of an implementation is shown according to the schematic of FIG. **13**. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. **14**, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. **15**, shown is a flow chart of a method of initializing the icons of FIG. **14**. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. **12** to **14**, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. **16** which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. **2**. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. **16** are performed by an install drop handler. During installation, verification is performed to determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. **17**, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. **15**), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. **17** which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. **13**. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. **16** and **17**, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim **1**, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim **2**, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

4. A method as defined in claim **1** further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim **2**, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim **2**, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim **2** further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim **1**, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim **2**, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim **2**, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim **2**, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim **11**, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim **1**.

13. A method as defined in claim **1** further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim **1** comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

a) running an instance of a service on a server;

b) determining which files are being used; and,

c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim **14** comprising the steps of:

assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;

installing the container on a server; and,

testing the applications and files within the container.

16. A method as defined in claim **1** comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim **1** further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes:

using a graphical user interface, associating a unique icon representing a service with an unique icon representing

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim **17** wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim **18**, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim **19**, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim **20** further comprising starting a distributed software application.

22. A method according to anyone of claims **20** further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim **17**, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim **17** further comprising creating a user account for the service.

25. A method as defined in claim **17**, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim **17** further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim **1**, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;

displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim **27** further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim **27** further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim **27** further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim **31**, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim **32**, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim **31**, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and,

forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

Exhibit 2

U.S. Patent No. 7,519,814 ("'814 Patent")

Accused Instrumentalities: Amazon's AWS End-of-Support Migration Program ("EMP"), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:	To the extent the preamble is limiting, Amazon practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed. *See* claim limitations below. *See also, e.g.*: AWS EMP for Windows Server enables application migration from legacy Windows Server systems to supported Windows operating systems. The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system based on your requirements, and migration to an AWS environment running a supported Windows Server version. When migrated to the new server environment, the application will run on the new OS as it would on the legacy operating system. The new package redirects application requests and does not include installation of any part of the legacy operating system. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023

Claim 1	Accused Instrumentalities
	EMP compatibility packages solve these challenges by resolving application dependencies on machine names, IP addresses, and ports that were created when the application was first installed on an out-of-support version of Windows Server. Network redirections are applied to packages so that the application can be migrated to a new server. EMP compatibility packages support Windows services common in Oracle Application Server, PeopleSoft, Tomcat, IBM WebSphere, and SQL Server 2005. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 AWS End-of-Support Migration Program (EMP) for Windows Server supports the following operating systems: • Windows Server 2019 (64-bit) • Windows Server 2016 (64-bit) • Windows Server 2012 R2 (64-bit) • Windows Server 2008 R2 (64-bit) • Windows Server 2008 (32-bit and 64-bit) • Windows Server 2003 SP2 (32-bit) The following system requirements must be met to migrate your application with AWS End-of-Support Migration Program (EMP) for Windows Server. • **.NET**. Microsoft .NET 4.0 Client Profile, or later. • **Memory**. As required by the packaged applications. Minimum 2 GB. • **Processor**. As required by the packaged applications. Two CPUs recommended. • **Disk space**. 10 GB required to create the snapshots. The size of the EMP package depends on the size of the application. It will be 10 to 50 percent larger than the application being packaged. https://docs.aws.amazon.com/emp/latest/userguide/emp-supported-os.html, Last accessed on July 13th, 2023 Kernel-mode drivers that are a different bitness than the target operating system. Device drivers are not virtualized with EMP and therefore must be compatible with the target operating system. Compatible drivers can be deployed with the package. For example, if you are moving to a 64-bit operating system, you must have a 64-bit driver that is compatible with the new operating system. https://docs.aws.amazon.com/emp/latest/userguide/emp-limitations.html, Last accessed on July 13th, 2023

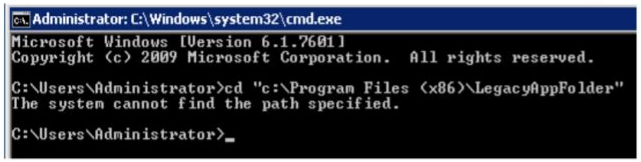
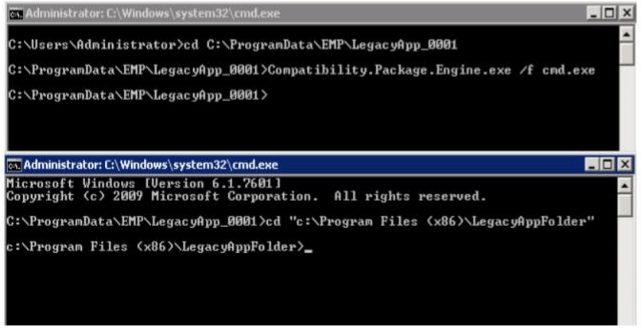
Claim 1	Accused Instrumentalities
	Phase 2: Compatibility Packaging AWS SMEs utilize the EMP Compatibility Packager, along with the information provided in Phase 1, to identify and package applications with libraries, files, and other dependencies. The resulting package resolves OS dependencies and enables the application to successfully run on new versions of Windows Server. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 **4.1.1 Application and Runtime Isolation** Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 Package an IIS-based application EMP supports packaging and migrating legacy IIS-based applications that run on Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 to the latest, supported versions of Windows Server running on AWS. https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023 Once the package has been launched by the start of a service, when an application shortcut or other entry point into the package is launched, another log file is created within the deployed package root folder called `RunWorkFlowLog.txt`. https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023

Claim 1	Accused Instrumentalities
	The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;	The method practiced by Amazon through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. *See, e.g.*: AWS EMP for Windows Server enables application migration from legacy Windows Server systems to supported Windows operating systems. The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system based on your requirements, and migration to an AWS environment running a supported Windows Server version. When migrated to the new server environment, the application will run on the new OS as it would on the legacy operating system. The new package redirects application requests and does not include installation of any part of the legacy operating system. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated) The application accesses data stored in a fixed location that is not available on the new OS version: the EMP engine redirects these requests to the appropriate location on the new version of the operating system. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)

Claim 1	Accused Instrumentalities
	The resulting package includes everything that the application needs to run on a modern operating system, including application files, runtimes, components, and deployment tools. The package does not include the legacy operating system, which means you never run any part of the legacy Windows Server version on the new Windows Server to which the application is upgraded. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated) The configuration of a package is defined in a series of XML files, which includes: • Configurations for Registry keys and values (AppRegistry.xml) • File, Registry, and Network redirections (Redirections.xml) • File Type Associations (FileAssociations.xml) • Shortcuts (Shortcuts.xml) • Executable programs (Programs.xml) • Environment Variables (EnvironmentVariables.xml) https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated) The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023

Claim 1	Accused Instrumentalities
	Kernel-mode drivers that are a different bitness than the target operating system. Device drivers are not virtualized with EMP and therefore must be compatible with the target operating system. Compatible drivers can be deployed with the package. For example, if you are moving to a 64-bit operating system, you must have a 64-bit driver that is compatible with the new operating system. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)
[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,	In the method practiced by Amazon through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. *See, e.g.*: AWS EMP for Windows Server enables application migration from legacy Windows Server systems to supported Windows operating systems. The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system based on your requirements, and migration to an AWS environment running a supported Windows Server version. When migrated to the new server environment, the application will run on the new OS as it would on the legacy operating system. The new package redirects application requests and does not include installation of any part of the legacy operating system. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated) The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)

Claim 1	Accused Instrumentalities
	Migrating legacy applications to newer, supported versions of Windows Server often requires refactoring, which is expensive and complex because of limited in-house expertise for such applications. With EMP for Windows Server, you no longer need to refactor your applications to ensure compatibility with newer versions of Windows Server. You now have the choice of using the tool as a self-service option at no cost or you can pay for the expert engagement to drive migration of your applications to AWS using the EMP tool. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)
[1c] the containers of application software excluding a kernel,	In the method practiced by Amazon through the Accused Instrumentalities, the containers of application software exclude a kernel. *See, e.g.*: **4.3.1 Application to Operating System Compatibility** EMP software runs legacy applications within a compatibility package on modern, up-to-date operating systems. The EMP compatibility package does not contain any parts of the legacy operating system. Instead, it intercepts the operating system requests and redirects and resolves them against the up-to-date host operating environment. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,	In the method practiced by Amazon through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server. *See, e.g.*: The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023

Claim 1	Accused Instrumentalities
	4.3.3 Services and Drivers EMP Compatibility packages support Windows Services out of the box. Drivers aren't captured in an EMP package but can be extracted and deployed locally using the EMP deployment script feature, as long as they are compatible with the target operating system. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 Use `cmd.exe` to check for the file in the local system. The system should not be able to find the file.  Run `cmd.exe` as a child process to the package engine. This should confirm that the `LegacyAppFolder` is available in the context of the EMP package.  https://docs.aws.amazon.com/emp/latest/userguide/emp-run-cmd-child.html, Last accessed on June 8th, 2023
[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local	In the method practiced by Amazon through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server. *See, e.g.*:

Claim 1	Accused Instrumentalities
system files that remain resident on the server,	**x64 (Engine Binaries)** — Contains the EMP runtime files for packages to be deployed on a 64-bit system. When packaging is performed on a 64-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 64-bit machine cannot be run on a 32-bit machine. **x86 (Engine Binaries)** — Contains the EMP runtime files for packages to be deployed on a 32-bit system. When packaging is performed on a 32-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 32-bit machine will run on a 64-bit machine, however, we recommend that you use the appropriate runtimes for the destination platform. This is automatically handled during the deployment process. https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023 Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023
[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,	In the method practiced by Amazon through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software. *See, e.g.*:

Claim 1	Accused Instrumentalities
	EMP software runs legacy applications within a compatibility package on modern, up-to-date operating systems. The EMP compatibility package does not contain any parts of the legacy operating system. Instead, it intercepts the operating system requests and redirects and resolves them against the up-to-date host operating environment. The EMP compatibility package permits the legacy application to run alongside other applications and/or other versions of the same application. For example, multiple versions of Microsoft Office can run simultaneously on the target operating system, or two incompatible 32-bit applications can run together, isolated from each other by the EMP compatibility package. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll. https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023
[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.	In the method practiced by Amazon through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system. *See, e.g.*:

Claim 1	Accused Instrumentalities
	This section describes the folders and files that are included in an EMP compatibility package. When the EMP compatibility packaging process is complete, the output of the package builder is called an EMP compatibility package. The package contains both the file and registry data of the packaged application, and the EMP binaries and configuration files that are required to deploy and run the packaged application. The EMP package, which is the product to deploy, is called the source package. The post-deployment package is called the deployed package. These packages are slightly different at the level of the root package folder. However, the packaged application content is the same in both packages. https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023 **x64 (Engine Binaries)** — Contains the EMP runtime files for packages to be deployed on a 64-bit system. When packaging is performed on a 64-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 64-bit machine cannot be run on a 32-bit machine. **x86 (Engine Binaries)** — Contains the EMP runtime files for packages to be deployed on a 32-bit system. When packaging is performed on a 32-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 32-bit machine will run on a 64-bit machine, however, we recommend that you use the appropriate runtimes for the destination platform. This is automatically handled during the deployment process. https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023

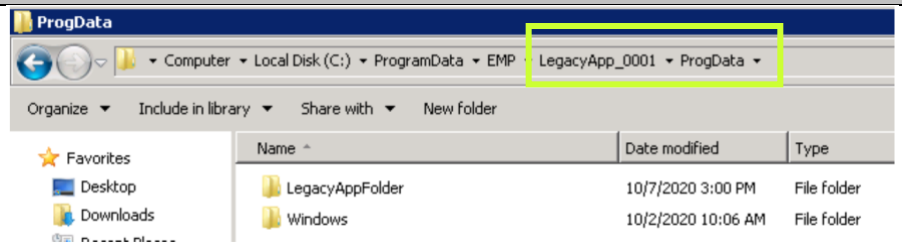
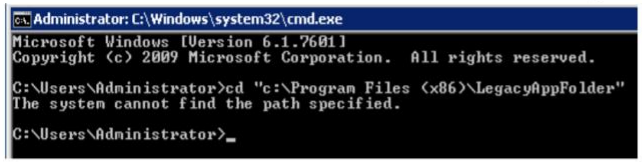
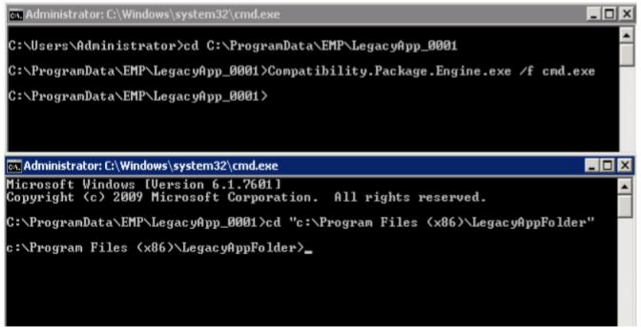
Claim 1	Accused Instrumentalities
	 https://docs.aws.amazon.com/emp/latest/userguide/emp-run-cmd-child.html, Last accessed on June 8th, 2023 (annotated) Use `cmd.exe` to check for the file in the local system. The system should not be able to find the file.  Run `cmd.exe` as a child process to the package engine. This should confirm that the `LegacyAppFolder` is available in the context of the EMP package.  https://docs.aws.amazon.com/emp/latest/userguide/emp-run-cmd-child.html, Last accessed on June 8th, 2023

Exhibit 3

US007784058B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.: US 7,784,058 B2**
(45) **Date of Patent: Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1293 days.

(21) Appl. No.: **10/946,536**

(22) Filed: **Sep. 21, 2004**

(65) **Prior Publication Data**

US 2005/0066303 A1 Mar. 24, 2005

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep. 22, 2003.

(51) **Int. Cl.**
G06F 9/22 (2006.01)

(52) **U.S. Cl.** **719/310**; 719/319

(58) **Field of Classification Search** 719/310, 719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,481,706	A *	1/1996	Peek	710/200
6,212,574	B1 *	4/2001	O'Rourke et al.	719/321
6,260,075	B1 *	7/2001	Cabrero et al.	719/310
2002/0004854	A1	1/2002	Hartley	
2002/0174215	A1	11/2002	Schaefer	709/224
2003/0101292	A1	5/2003	Fisher	
2004/0025165	A1 *	2/2004	Desoli et al.	719/310

FOREIGN PATENT DOCUMENTS

WO	WO 02/06941	A	1/2002
WO	WO 2006/039181	A	4/2006

OTHER PUBLICATIONS

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

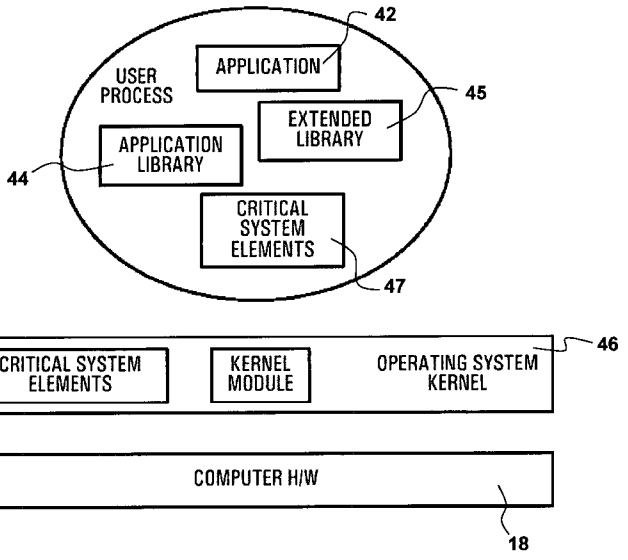
* cited by examiner

Primary Examiner—Hyung S Sough
Assistant Examiner—Syed Roni
(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A. Attorneys at Law

(57) **ABSTRACT**

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

18 Claims, 7 Drawing Sheets



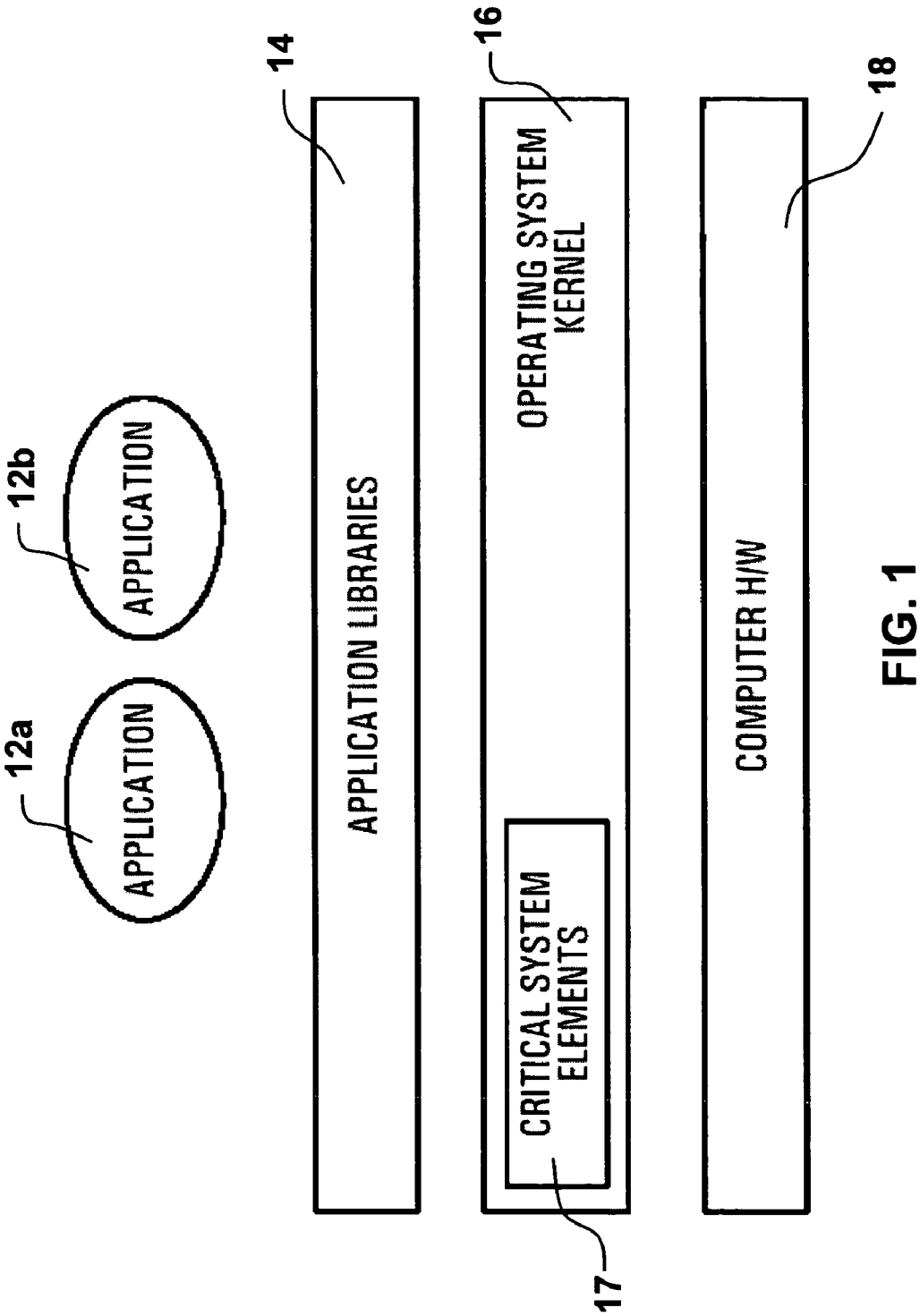


FIG. 1
Prior Art

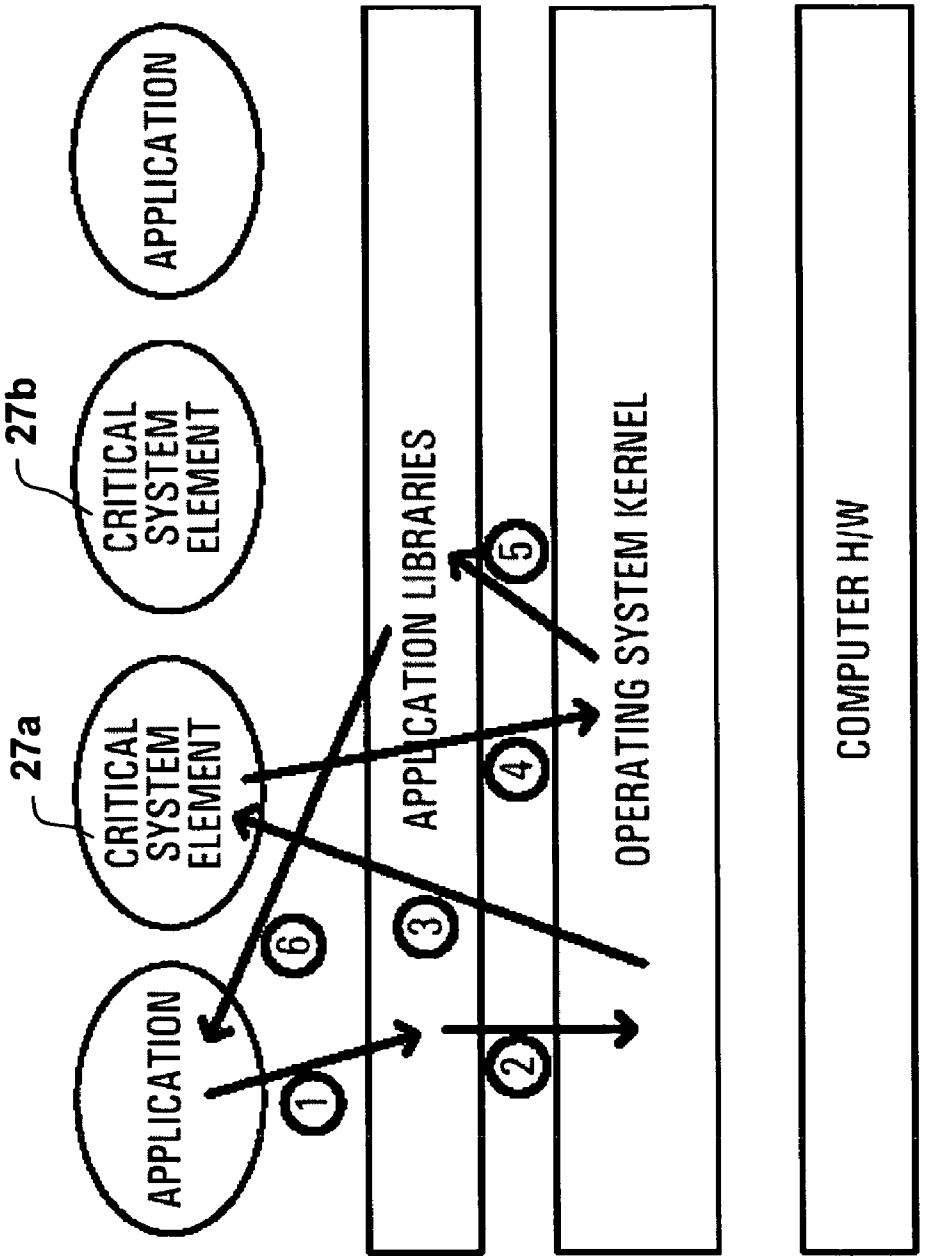


FIG. 2a
Prior Art

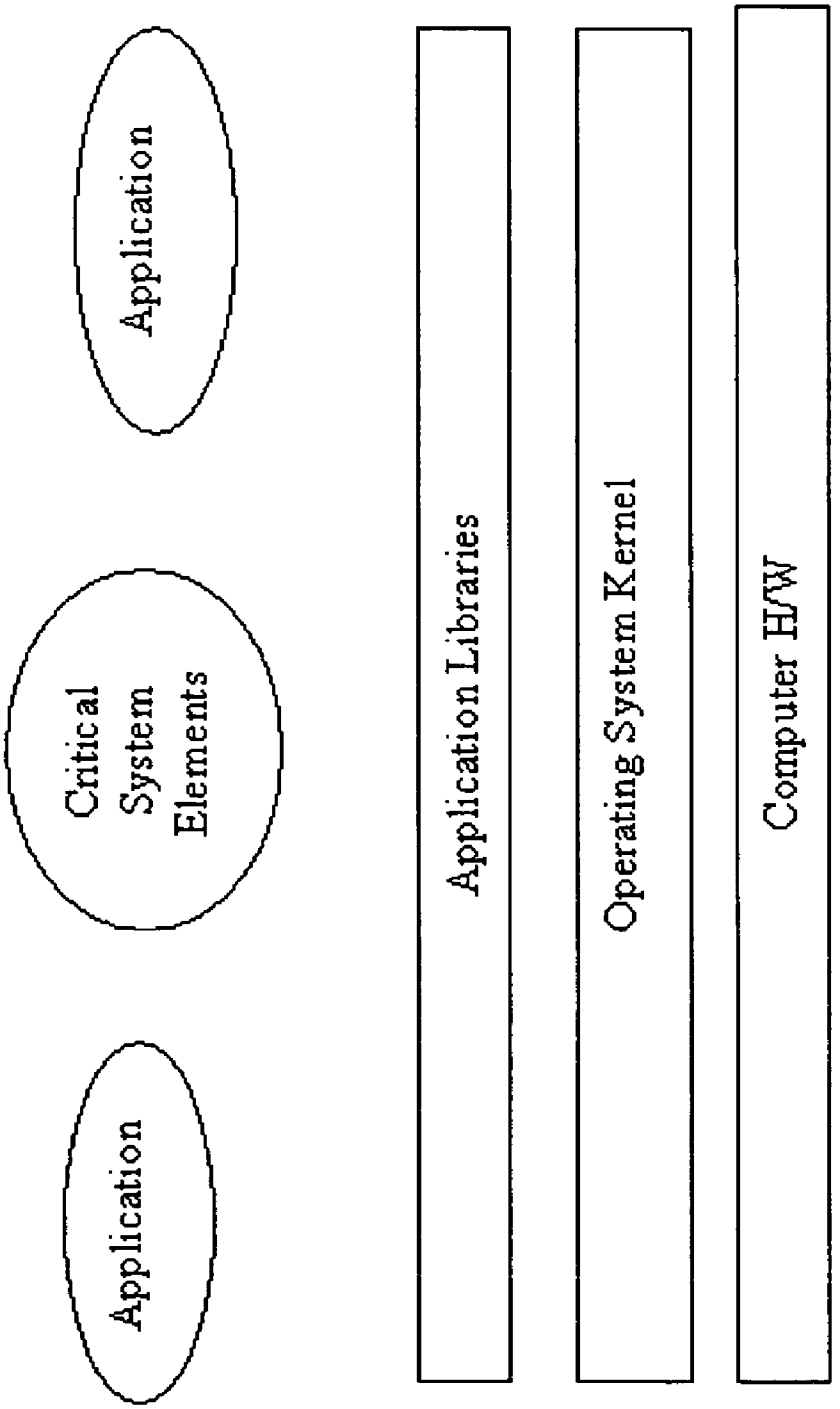


FIG. 2b
Prior Art

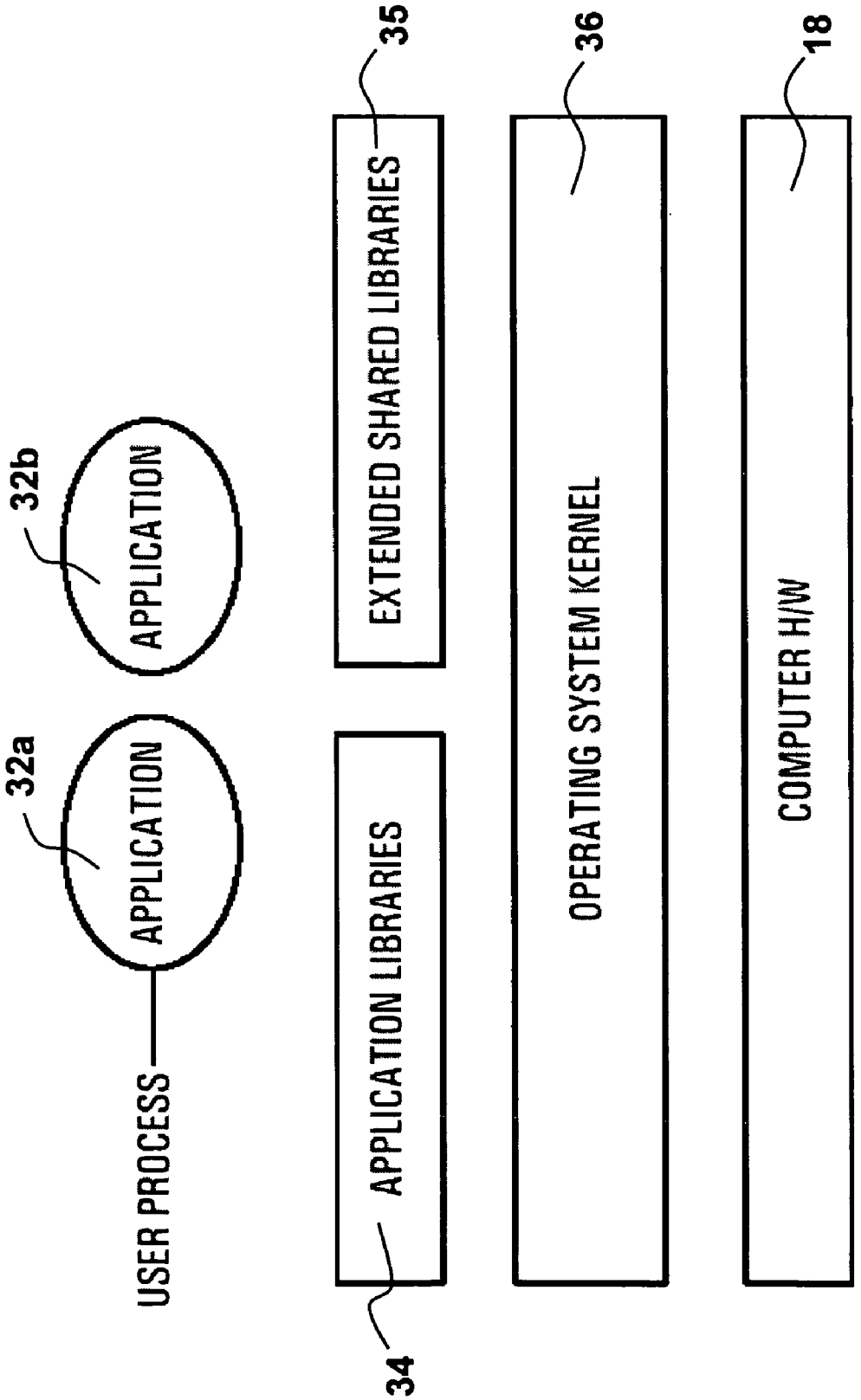

35
36
18
32b
APPLICATION
32a
APPLICATION
USER PROCESS
APPLICATION LIBRARIES
EXTENDED SHARED LIBRARIES
OPERATING SYSTEM KERNEL
COMPUTER H/W
34

FIG. 3

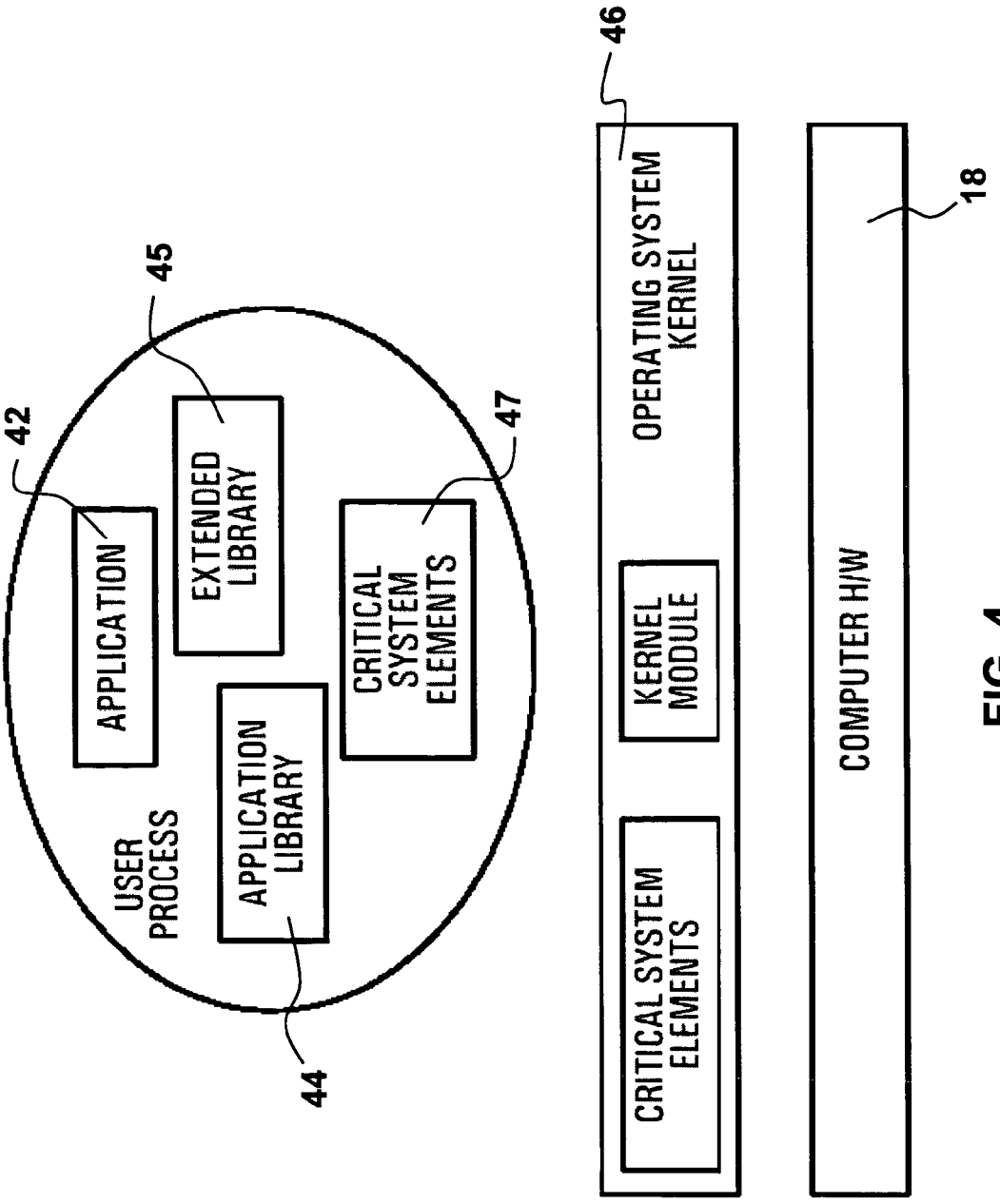

46
OPERATING SYSTEM KERNEL
KERNEL MODULE
CRITICAL SYSTEM ELEMENTS
COMPUTER H/W
18
45
42
47
APPLICATION
EXTENDED LIBRARY
CRITICAL SYSTEM ELEMENTS
APPLICATION LIBRARY
USER PROCESS
44

FIG. 4

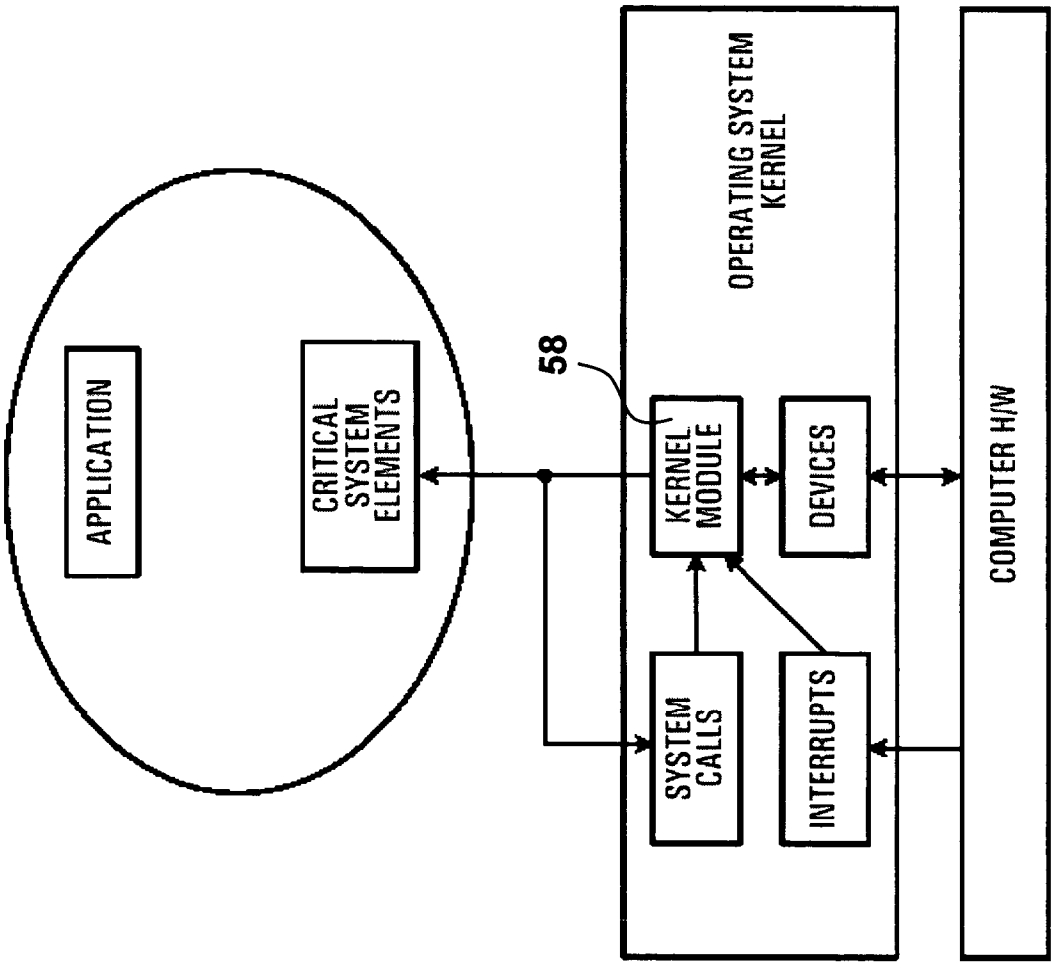

APPLICATION
CRITICAL SYSTEM ELEMENTS
58
OPERATING SYSTEM KERNEL
KERNEL MODULE
DEVICES
SYSTEM CALLS
INTERRUPTS
COMPUTER H/W

FIG. 5

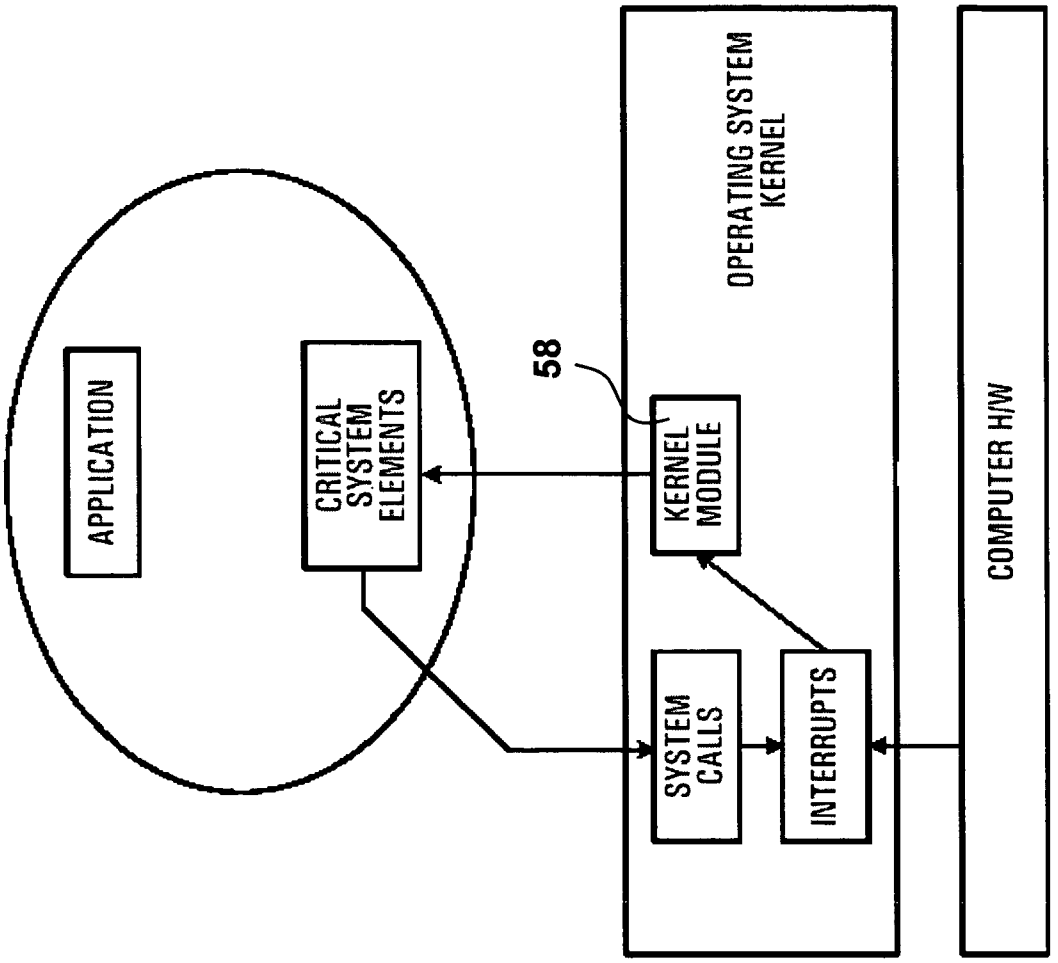

APPLICATION
CRITICAL SYSTEM ELEMENTS
58
KERNEL MODULE
OPERATING SYSTEM KERNEL
SYSTEM CALLS
INTERRUPTS
COMPUTER H/W

FIG. 6

COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled "User Mode Critical System Element as Shared Libs", which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.
2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. **1** is an architectural view of the traditional monolithic prior art operating system;

FIG. **2***a* is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. **2***b* is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. **3** is an architectural view of an embodiment of the invention;

FIG. **4** is a functional view showing how critical system elements exist in the same context as an application;

FIG. **5** is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. **6** shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TPC/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality
2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided
2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. **1** shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. **1** to make use of a critical system element **17** in the kernel **16**, the application **12***a* or **12***b*

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12***a*, **12***b* makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12***a*, **12***b* in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2***a* shows a system architecture where critical system elements **27***a*, **27***b* execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2***a* is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2***a* as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2***b* is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2***a* and FIG. **2***b* is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2***b* is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the case with a multi-server architecture depicted in FIG. **2***a*. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32***a*, **32***b*, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45**,**47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. **3** shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. **4** shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. **4** the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. **4**. FIG. **5** shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. **6** a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:

a) a processor;

b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,

c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,

ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim **1**, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim **1** wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim **1** wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim **1** wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim **5** wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim **6** wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim **7** wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

9. A computing system according to claim **2**, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim **2** wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim **2** wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim **1**, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim **10** wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim **13** wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim **5** wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim **1** wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim **2** wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim **2** wherein SLCSEs are not copies of OSCSEs.

* * * * *

Exhibit 4

U.S. Patent No. 7,519,814 ("'814 Patent")

Accused Instrumentalities: the Amazon Elastic Container Service ("ECS"), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

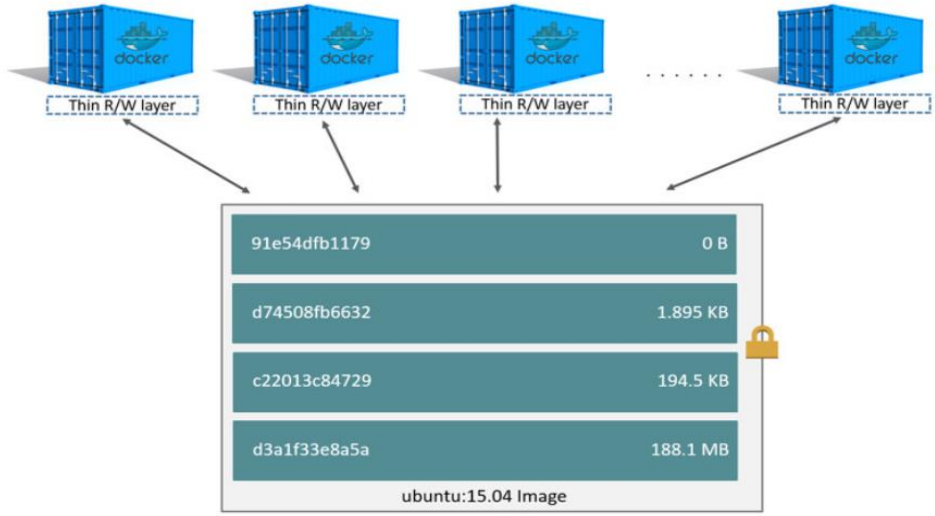
Claim 1	Accused Instrumentalities
[1pre] 1. A computing system for executing a plurality of software applications comprising:	To the extent the preamble is limiting, each Accused Instrumentality comprise or constitute a computing system for executing a plurality of software applications as claimed. *See* claim limitations below. *See also, e.g.*: Amazon ECS capacity Amazon ECS capacity is the infrastructure where your containers run. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 There are three layers in Amazon ECS: • Capacity - The infrastructure where your containers run • Controller - Deploy and manage your applications that run on the containers • Provisioning - The tools that you can use to interface with the scheduler to deploy and manage your applications and containers https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023

Claim 1	Accused Instrumentalities
	To deploy applications on Amazon ECS, your application components must be configured to run in *containers*. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an *image*. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a *registry* such as Amazon ECR where they can be downloaded from. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023
[1a] a) a processor;	Each Accused Instrumentality comprises a processor. *See, e.g.*: Amazon ECS supports using 64-bit ARM applications. You can run your applications on the platform that's powered by AWS Graviton2 processors,. It's suitable for a wide variety of workloads. This includes workloads such as application servers, micro-services, high-performance computing, CPU-based machine learning inference, video encoding, electronic design automation, gaming, open-source databases, and in-memory caches. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 (annotated)
[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,	Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor. *See, e.g.*:

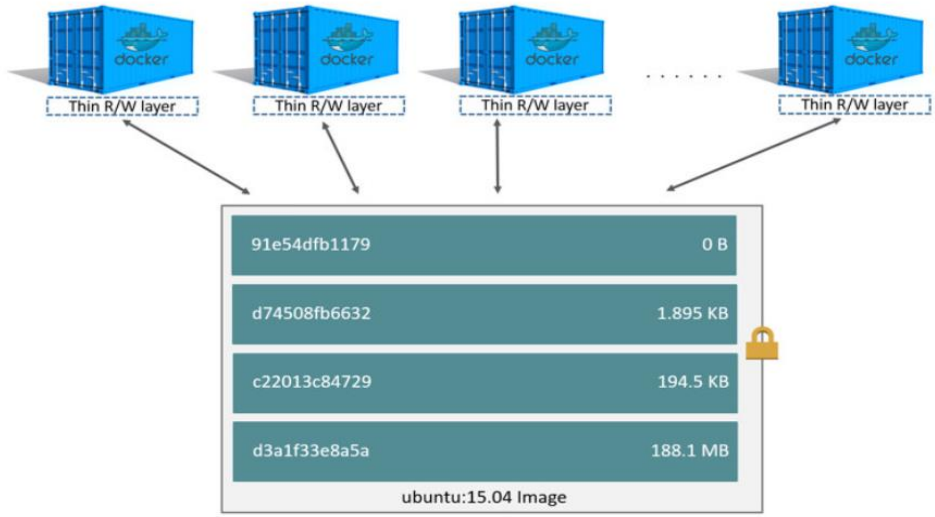
Claim 1	Accused Instrumentalities
	An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into an Amazon ECS cluster. When you run tasks with Amazon ECS using the EC2 launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances. **Note** Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply. The following Linux container instance operating systems are available: • Amazon Linux: This is a general purpose operating system. • Bottlerocket: This is an operating system that is optimized for container workloads and that has a focus on security. It does not include a package manager and is immutable by default. For information about the security features and guidance, see Security Features and Security Guidance on the GitHub website. An Amazon ECS container instance specification consists of the following components: **Required** • A modern Linux distribution running at least version 3.10 of the Linux kernel. • The Amazon ECS container agent (preferably the latest version). For more information, see Amazon ECS container agent (p. 357). https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 (annotated)
[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and	Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode. *See, e.g.*:

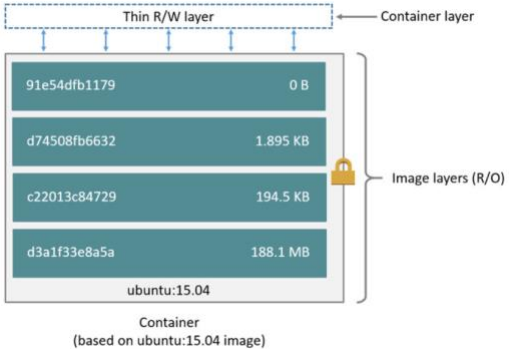
Claim 1	Accused Instrumentalities
	To deploy applications on Amazon ECS, your application components must be configured to run in *containers*. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an *image*. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a *registry* such as Amazon ECR where they can be downloaded from. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see Deploying Docker containers on Amazon ECS. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 A Docker image is a read-only template that defines your container. The image contains the code that will run including any definitions for any libraries and dependancies your code needs. A Docker container is an instantiated (running) Docker image. AWS provides Amazon Elastic Container Registry (ECR), an image registry for storing and quickly retrieving Docker images. https://aws.amazon.com/docker/#, Last accessed on June 14, 2023
[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of	In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications. For example, a Docker base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and

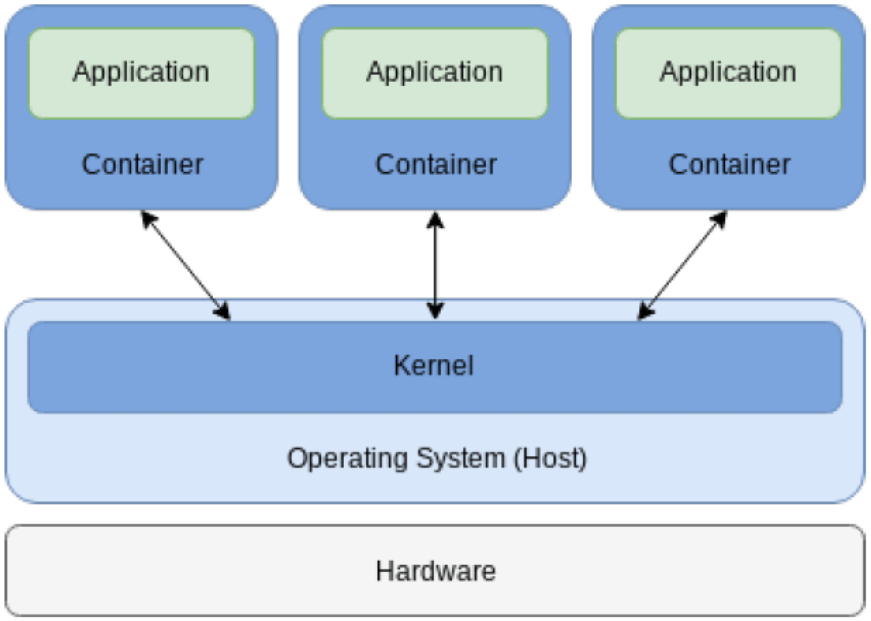
Claim 1	Accused Instrumentalities
the one or more of the plurality of software applications,	dependencies, including system libraries or shared library critical system elements (SLCSEs). When the container runs the image, it creates a runtime instance of that container image. *See, e.g.*: Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container. https://www.techtarget.com/searchitoperations/definition/Docker-image, Last accessed on June 14, 2023

Claim 1	Accused Instrumentalities
	Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.  https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023
[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software	In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function. When a Docker image is used to create a container, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications

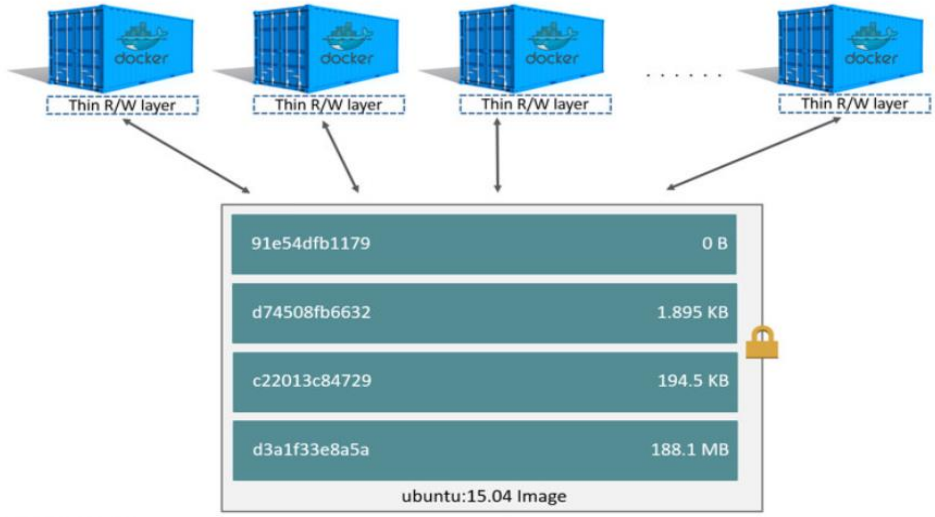
Claim 1	Accused Instrumentalities
applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and	running in their respective containers. The docker image includes essential system files, libraries, and dependencies required to run the software application within the container. The Docker containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it operates in isolation and runs its own instance of the software application without sharing resources or critical system elements with other containers. This ensures that each container has its own isolated context. Docker containers can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same or a different function. *See, e.g.*: Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see Deploying Docker containers on Amazon ECS. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 A Docker image is a read-only template that defines your container. The image contains the code that will run including any definitions for any libraries and dependancies your code needs. A Docker container is an instantiated (running) Docker image. AWS provides Amazon Elastic Container Registry (ECR), an image registry for storing and quickly retrieving Docker images. https://aws.amazon.com/docker/#, Last accessed on June 14, 2023

Claim 1	Accused Instrumentalities
	Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container. https://www.techtarget.com/searchitoperations/definition/Docker-image, Last accessed on June 14, 2023 Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.  https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023

Claim 1	Accused Instrumentalities
	By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine. https://docs.docker.com/get-started/overview/, Last accessed on June 14, 2023 The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023

Claim 1	Accused Instrumentalities
	 https://www.researchgate.net/figure/Docker-container-architecture_fig1_333235708, Last accessed on June 14, 2023
[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a	In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously. For example, In Docker, each container operates independently, and a Docker base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When a Docker image is used to create a container in ECS, an instance of the SLCSE is provided to a software application. Therefore, different instances of the SLCSE are

Claim 1	Accused Instrumentalities
second instance of the SLCSE simultaneously.	provided to different applications for performing either a same or a different function, simultaneously. *See, e.g.*: Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see Deploying Docker containers on Amazon ECS. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 A Docker image is a read-only template that defines your container. The image contains the code that will run including any definitions for any libraries and dependancies your code needs. A Docker container is an instantiated (running) Docker image. AWS provides Amazon Elastic Container Registry (ECR), an image registry for storing and quickly retrieving Docker images. https://aws.amazon.com/docker/#, Last accessed on June 14, 2023 Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container. https://www.techtarget.com/searchitoperations/definition/Docker-image, Last accessed on June 14, 2023

Claim 1	Accused Instrumentalities
	Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.  https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023

JS 44 (Rev. 10/20)

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. *(SEE INSTRUCTIONS ON NEXT PAGE OF THIS FORM.)*

I. (a) PLAINTIFFS

VIRTAMOVE, CORP.

(b) County of Residence of First Listed Plaintiff _____
(EXCEPT IN U.S. PLAINTIFF CASES)

(c) Attorneys *(Firm Name, Address, and Telephone Number)*

Russ August & Kabat,12424 Wilshire Boulevard, 12th Floor, Los Angeles, California 90025,Tel.: (310) 826-7474

DEFENDANTS

AMAZON.COM, INC.; AMAZON.COM SERVICES LLC; AND AMAZON WEB SERVICES, INC.

County of Residence of First Listed Defendant _____
(IN U.S. PLAINTIFF CASES ONLY)

NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF THE TRACT OF LAND INVOLVED.

Attorneys *(If Known)*

II. BASIS OF JURISDICTION *(Place an "X" in One Box Only)*

- ☐ 1 U.S. Government Plaintiff
- ☒ 3 Federal Question *(U.S. Government Not a Party)*
- ☐ 2 U.S. Government Defendant
- ☐ 4 Diversity *(Indicate Citizenship of Parties in Item III)*

III. CITIZENSHIP OF PRINCIPAL PARTIES *(Place an "X" in One Box for Plaintiff and One Box for Defendant)*

(For Diversity Cases Only)

	PTF	DEF		PTF	DEF
Citizen of This State	☐ 1	☐ 1	Incorporated *or* Principal Place of Business In This State	☐ 4	☐ 4
Citizen of Another State	☐ 2	☐ 2	Incorporated *and* Principal Place of Business In Another State	☐ 5	☐ 5
Citizen or Subject of a Foreign Country	☐ 3	☐ 3	Foreign Nation	☐ 6	☐ 6

IV. NATURE OF SUIT *(Place an "X" in One Box Only)*

Click here for: Nature of Suit Code Descriptions.

CONTRACT
- ☐ 110 Insurance
- ☐ 120 Marine
- ☐ 130 Miller Act
- ☐ 140 Negotiable Instrument
- ☐ 150 Recovery of Overpayment & Enforcement of Judgment
- ☐ 151 Medicare Act
- ☐ 152 Recovery of Defaulted Student Loans (Excludes Veterans)
- ☐ 153 Recovery of Overpayment of Veteran's Benefits
- ☐ 160 Stockholders' Suits
- ☐ 190 Other Contract
- ☐ 195 Contract Product Liability
- ☐ 196 Franchise

REAL PROPERTY
- ☐ 210 Land Condemnation
- ☐ 220 Foreclosure
- ☐ 230 Rent Lease & Ejectment
- ☐ 240 Torts to Land
- ☐ 245 Tort Product Liability
- ☐ 290 All Other Real Property

TORTS

PERSONAL INJURY
- ☐ 310 Airplane
- ☐ 315 Airplane Product Liability
- ☐ 320 Assault, Libel & Slander
- ☐ 330 Federal Employers' Liability
- ☐ 340 Marine
- ☐ 345 Marine Product Liability
- ☐ 350 Motor Vehicle
- ☐ 355 Motor Vehicle Product Liability
- ☐ 360 Other Personal Injury
- ☐ 362 Personal Injury - Medical Malpractice

PERSONAL INJURY
- ☐ 365 Personal Injury - Product Liability
- ☐ 367 Health Care/ Pharmaceutical Personal Injury Product Liability
- ☐ 368 Asbestos Personal Injury Product Liability

PERSONAL PROPERTY
- ☐ 370 Other Fraud
- ☐ 371 Truth in Lending
- ☐ 380 Other Personal Property Damage
- ☐ 385 Property Damage Product Liability

CIVIL RIGHTS
- ☐ 440 Other Civil Rights
- ☐ 441 Voting
- ☐ 442 Employment
- ☐ 443 Housing/ Accommodations
- ☐ 445 Amer. w/Disabilities - Employment
- ☐ 446 Amer. w/Disabilities - Other
- ☐ 448 Education

PRISONER PETITIONS

Habeas Corpus:
- ☐ 463 Alien Detainee
- ☐ 510 Motions to Vacate Sentence
- ☐ 530 General
- ☐ 535 Death Penalty

Other:
- ☐ 540 Mandamus & Other
- ☐ 550 Civil Rights
- ☐ 555 Prison Condition
- ☐ 560 Civil Detainee - Conditions of Confinement

FORFEITURE/PENALTY
- ☐ 625 Drug Related Seizure of Property 21 USC 881
- ☐ 690 Other

LABOR
- ☐ 710 Fair Labor Standards Act
- ☐ 720 Labor/Management Relations
- ☐ 740 Railway Labor Act
- ☐ 751 Family and Medical Leave Act
- ☐ 790 Other Labor Litigation
- ☐ 791 Employee Retirement Income Security Act

IMMIGRATION
- ☐ 462 Naturalization Application
- ☐ 465 Other Immigration Actions

BANKRUPTCY
- ☐ 422 Appeal 28 USC 158
- ☐ 423 Withdrawal 28 USC 157

PROPERTY RIGHTS
- ☐ 820 Copyrights
- ☒ 830 Patent
- ☐ 835 Patent - Abbreviated New Drug Application
- ☐ 840 Trademark
- ☐ 880 Defend Trade Secrets Act of 2016

SOCIAL SECURITY
- ☐ 861 HIA (1395ff)
- ☐ 862 Black Lung (923)
- ☐ 863 DIWC/DIWW (405(g))
- ☐ 864 SSID Title XVI
- ☐ 865 RSI (405(g))

FEDERAL TAX SUITS
- ☐ 870 Taxes (U.S. Plaintiff or Defendant)
- ☐ 871 IRS—Third Party 26 USC 7609

OTHER STATUTES
- ☐ 375 False Claims Act
- ☐ 376 Qui Tam (31 USC 3729(a))
- ☐ 400 State Reapportionment
- ☐ 410 Antitrust
- ☐ 430 Banks and Banking
- ☐ 450 Commerce
- ☐ 460 Deportation
- ☐ 470 Racketeer Influenced and Corrupt Organizations
- ☐ 480 Consumer Credit (15 USC 1681 or 1692)
- ☐ 485 Telephone Consumer Protection Act
- ☐ 490 Cable/Sat TV
- ☐ 850 Securities/Commodities/ Exchange
- ☐ 890 Other Statutory Actions
- ☐ 891 Agricultural Acts
- ☐ 893 Environmental Matters
- ☐ 895 Freedom of Information Act
- ☐ 896 Arbitration
- ☐ 899 Administrative Procedure Act/Review or Appeal of Agency Decision
- ☐ 950 Constitutionality of State Statutes

V. ORIGIN *(Place an "X" in One Box Only)*

☒ 1 Original Proceeding ☐ 2 Removed from State Court ☐ 3 Remanded from Appellate Court ☐ 4 Reinstated or Reopened ☐ 5 Transferred from Another District *(specify)* ☐ 6 Multidistrict Litigation - Transfer ☐ 8 Multidistrict Litigation - Direct File

VI. CAUSE OF ACTION

Cite the U.S. Civil Statute under which you are filing ***(Do not cite jurisdictional statutes unless diversity)***:
35 U.S.C. § 1 et seq.

Brief description of cause:
Patent Infringement

VII. REQUESTED IN COMPLAINT:

☐ CHECK IF THIS IS A **CLASS ACTION** UNDER RULE 23, F.R.Cv.P.

DEMAND $

CHECK YES only if demanded in complaint:

JURY DEMAND: ☒ Yes ☐ No

VIII. RELATED CASE(S) IF ANY

(See instructions): JUDGE _____ DOCKET NUMBER _____

DATE
1/26/2023

SIGNATURE OF ATTORNEY OF RECORD
/s/ Reza Mirzaie

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

INSTRUCTIONS FOR ATTORNEYS COMPLETING CIVIL COVER SHEET FORM JS 44

Authority For Civil Cover Sheet

The JS 44 civil cover sheet and the information contained herein neither replaces nor supplements the filings and service of pleading or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. Consequently, a civil cover sheet is submitted to the Clerk of Court for each civil complaint filed. The attorney filing a case should complete the form as follows:

I.(a) Plaintiffs-Defendants. Enter names (last, first, middle initial) of plaintiff and defendant. If the plaintiff or defendant is a government agency, use only the full name or standard abbreviations. If the plaintiff or defendant is an official within a government agency, identify first the agency and then the official, giving both name and title.

(b) County of Residence. For each civil case filed, except U.S. plaintiff cases, enter the name of the county where the first listed plaintiff resides at the time of filing. In U.S. plaintiff cases, enter the name of the county in which the first listed defendant resides at the time of filing. (NOTE: In land condemnation cases, the county of residence of the "defendant" is the location of the tract of land involved.)

(c) Attorneys. Enter the firm name, address, telephone number, and attorney of record. If there are several attorneys, list them on an attachment, noting in this section "(see attachment)".

II. Jurisdiction. The basis of jurisdiction is set forth under Rule 8(a), F.R.Cv.P., which requires that jurisdictions be shown in pleadings. Place an "X" in one of the boxes. If there is more than one basis of jurisdiction, precedence is given in the order shown below.
United States plaintiff. (1) Jurisdiction based on 28 U.S.C. 1345 and 1348. Suits by agencies and officers of the United States are included here.
United States defendant. (2) When the plaintiff is suing the United States, its officers or agencies, place an "X" in this box.
Federal question. (3) This refers to suits under 28 U.S.C. 1331, where jurisdiction arises under the Constitution of the United States, an amendment to the Constitution, an act of Congress or a treaty of the United States. In cases where the U.S. is a party, the U.S. plaintiff or defendant code takes precedence, and box 1 or 2 should be marked.
Diversity of citizenship. (4) This refers to suits under 28 U.S.C. 1332, where parties are citizens of different states. When Box 4 is checked, the citizenship of the different parties must be checked. (See Section III below; **NOTE: federal question actions take precedence over diversity cases.**)

III. Residence (citizenship) of Principal Parties. This section of the JS 44 is to be completed if diversity of citizenship was indicated above. Mark this section for each principal party.

IV. Nature of Suit. Place an "X" in the appropriate box. If there are multiple nature of suit codes associated with the case, pick the nature of suit code that is most applicable. Click here for: Nature of Suit Code Descriptions.

V. Origin. Place an "X" in one of the seven boxes.
Original Proceedings. (1) Cases which originate in the United States district courts.
Removed from State Court. (2) Proceedings initiated in state courts may be removed to the district courts under Title 28 U.S.C., Section 1441.
Remanded from Appellate Court. (3) Check this box for cases remanded to the district court for further action. Use the date of remand as the filing date.
Reinstated or Reopened. (4) Check this box for cases reinstated or reopened in the district court. Use the reopening date as the filing date.
Transferred from Another District. (5) For cases transferred under Title 28 U.S.C. Section 1404(a). Do not use this for within district transfers or multidistrict litigation transfers.
Multidistrict Litigation – Transfer. (6) Check this box when a multidistrict case is transferred into the district under authority of Title 28 U.S.C. Section 1407.
Multidistrict Litigation – Direct File. (8) Check this box when a multidistrict case is filed in the same district as the Master MDL docket.
PLEASE NOTE THAT THERE IS NOT AN ORIGIN CODE 7. Origin Code 7 was used for historical records and is no longer relevant due to changes in statue.

VI. Cause of Action. Report the civil statute directly related to the cause of action and give a brief description of the cause. **Do not cite jurisdictional statutes unless diversity.** Example: U.S. Civil Statute: 47 USC 553 Brief Description: Unauthorized reception of cable service.

VII. Requested in Complaint. Class Action. Place an "X" in this box if you are filing a class action under Rule 23, F.R.Cv.P.
Demand. In this space enter the actual dollar amount being demanded or indicate other demand, such as a preliminary injunction.
Jury Demand. Check the appropriate box to indicate whether or not a jury is being demanded.

VIII. Related Cases. This section of the JS 44 is used to reference related pending cases, if any. If there are related pending cases, insert the docket numbers and the corresponding judge names for such cases.

Date and Attorney Signature. Date and sign the civil cover sheet.